Diploma Project

Manage Telecommunication equipment using Web Services

Acronym: TELECOM-WS
Number: D09T02

Date: 25.05.09 – 10.07.09

Professeurs:
Philippe Joye
François Buntschu

Students:
Thierry Kiki
David Schneider

Mandatory:
Daniel Gachet

Expert:
Nicolas Mayencourt

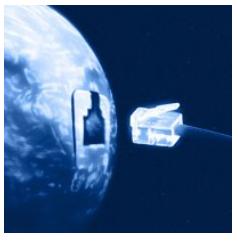



Table of contents

1 Introduction

Abstract

This first chapter introduces you into the Bachelor Project of Thierry Kiki and David Schneider. Necessary definitions and explications for the understanding of the report are provided here.

1.1 Definitions

MileGate

MO:	Managed Object
MOM:	Managed Object Model
moType:	Managed Object Type
MF:	Management Function
ADF:	AccesPoint (MO) – definition file
SFD:	Proprietary structure file of MileGate
MCST:	MileGate Configuration Software Tool
KOAP:	KEYMILE Object Access Protocol

Web Service

XML:	Extensible Markup Language (W3C)
SOAP:	Simple Object Access Protocol (W3C)
Web Services:	W3C recommendation
GUI:	Graphical User Interface

Others

ECLI:	Embedded Comand Line Interface
HMI:	Human-Machine Interface
MMI:	Machine-Machine Interface

1.2 Project

This section describes the project and introduces the equipment. We defined the objectives we planed to reach at the initiation phase of the project. The distribution of tasks between the participators of the project will also be defined here.

1.2.1 Introduction

The company KEYMILE wishes a utility to manage its next generation telecommunication equipment with a system using web services. Actually, the management of the object model is performed either over an embedded command line (ECLI), syslog, SNMP or with the exchange of proprietary XML messages (KM-KOAP). The aim of this project is to find standardized solutions using web services (MMI) or to offer access via a web browser (HMI).

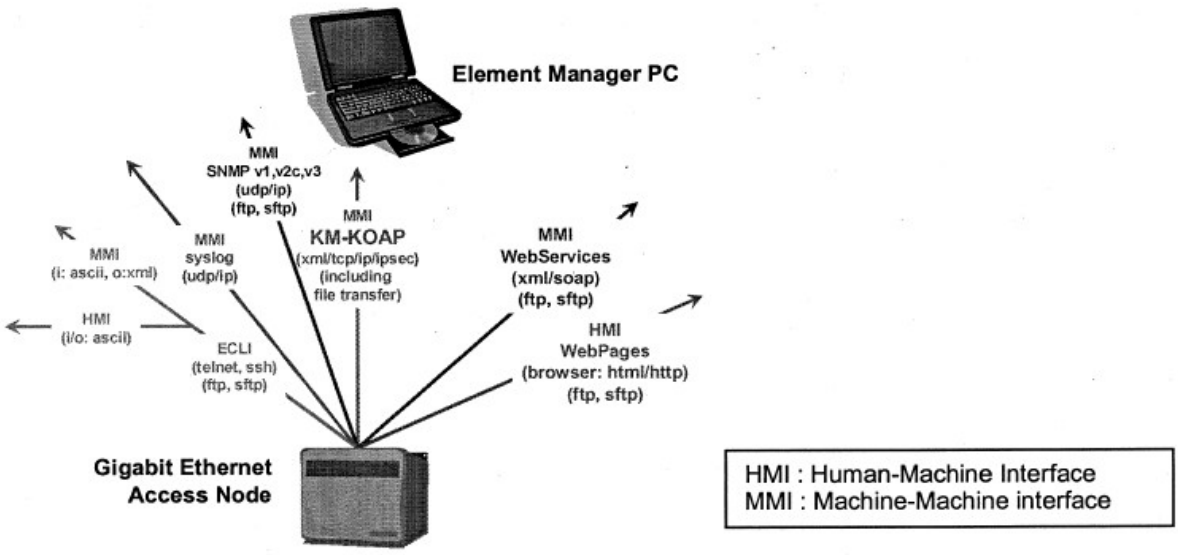


Illustration 1: MILEGATE management interfaces

1.2.2 Description of project

As this project needs to be adapted to the existing system, we need to respect a few constraints.
In the following image, the relations between the KEYMILE file describing the internal object model (SFD, XML) and the AccessPoint Definition File ADF (proprietary, XML).

1.2.2.1 Actual management system

At the moment, the management system uses ADF which is a collection of SFD and describes one single unit in the MileGate. The core unit and access point of the MileGate is in slot 11.

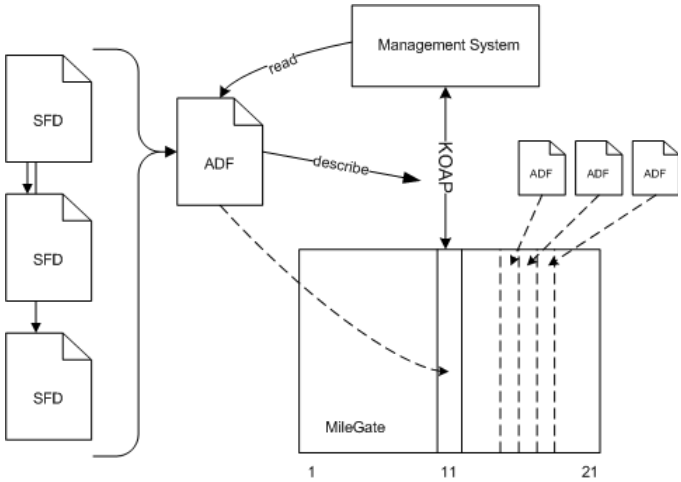


Illustration 2: Existing system

If we represent the actual communication more in detail, we see how the existing management utilities access the MileGate Object Model.

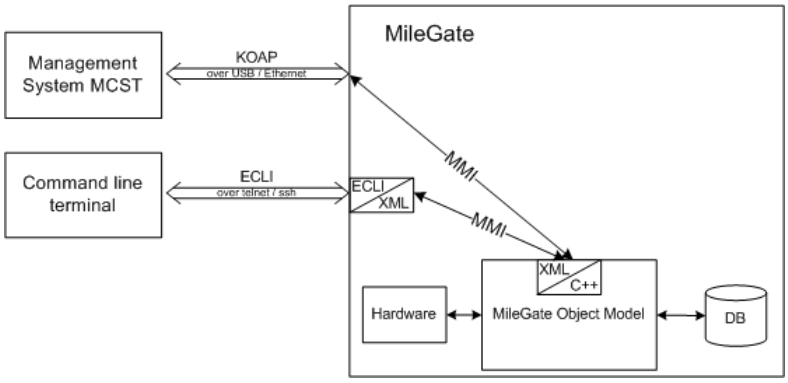


Illustration 3: Existing access methods

1.2.2.2 Work to perform

We have two new approaches for accessing the MileGate Object Model. The tasks to perform are represented in red.

1.2.2.2.1 Machine-Machine Interface (MMI)

The Web Service Description (WSDL file) which would finally be created will be the input for the client framework. The framework will generate code (for example Java, C, C++, Perl, Pyton, PHP, ..) automatically according to the constraints defined in the web service description.
The messages of the type SOAP (transported over HTTP) are treated within the embedded HTTP server and afterward transformed from SOAP-XML into the proprietary XML format which is the only interface of the MileGate's embedded system.

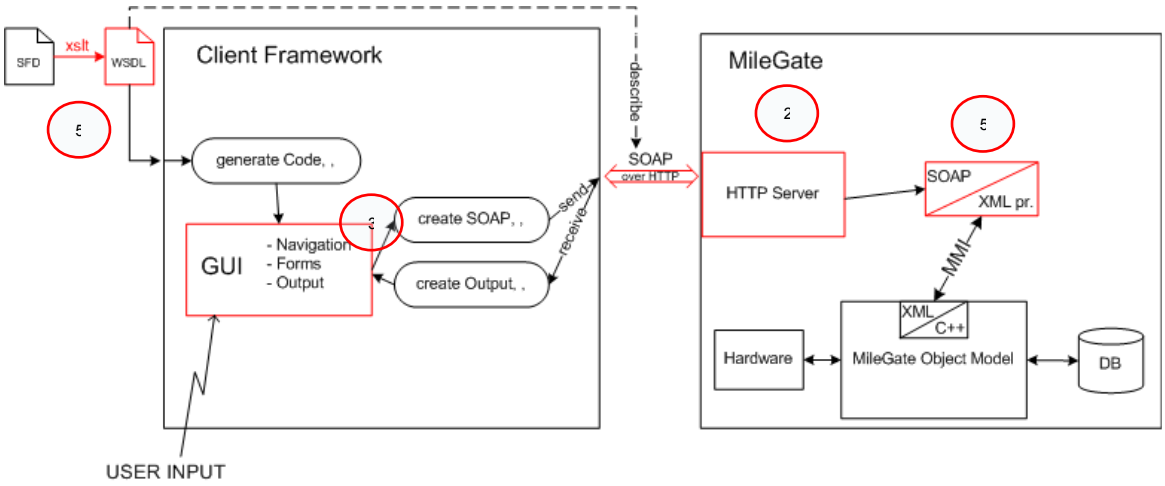


Illustration 4: Approach with web service

1.2.2.2.2 Human-Machine Interface (HMI)

To be accessible by humans, the MileGate should provide HTML files generated at runtime. Therefore, we connect the HTTP Server not with the Client framework but with a web browser.

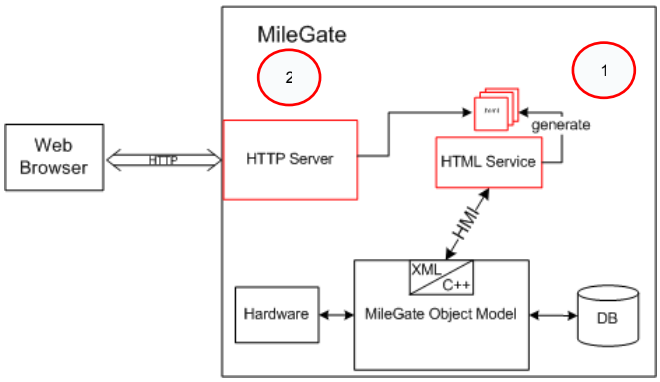


Illustration 5: Approach with generation of HTML files

For both approaches the MileGate Object Model manages the call of the C++ routines from the proprietary XML messages. In addition, it communicates with the hardware and affects storage/request of data.

1.2.3 Objectives

The sections contains the definition of the objectives. The numbers refer to the section "1.2.2.2 Work to perform".

1.2.3.1 Side issue

Find a good way to generate on the fly HTML pages within the MilGate which providing a web browser access. 1

Survey and evaluate different embedded HTTP servers running on Linux and/or VxWorks for the MilGate. 2

1.2.3.2 Main issue

Survey and evaluate different client frameworks and describe their compatibilities with the Web Services. 3

Describe the flow of information from the KEYMILE files which describes the internal structure through the embedded HTTP Server to the MileGate 4

Define the web service and the necessary transformation. 5

Implement a prototype using the web service (MMI). 6

1.2.4 Distribution of Tasks

A strict division of the responsibilities was demanded. Even though the project was difficult to subdivide at the initiation phase, we split the tasks after some discussions as followed:

Thierry Kiki has to:

- Survey and evaluate different embedded HTTP servers running on Linux and/or VxWorks for the MilGate.
- Survey and evaluate different client Frameworks and describe the tools offer by that framework to consume web services.
- Survey the interoperability between web services and description of WS-I tools

The resulting chapter are "3.1 Embedded HTTP server" and "6. Web Service Tools"

David Schneider studies the feasibility for HTML generation within the MileGate and recommends a implementation.
Afterward a Web Service will be described using WSDL and with it the automatic transformation to exchanged SOAP messages. The transformation from description files to WSDL and from SOAP to the proprietary XML format have to be adapted and described.
The resulting chapter are "2. Introduction into Web Services", "3.2 HTML generation service" "4. Web Service Description", "5. Web Service Concepts" and the Annexes XML, XSLT and HTML Service

The initial planing (Gant diagram) is represented as Annexe:Planing

1.3 What is MileGate?

MileGate is an IP-based multi-service next-generation access platform that can support you in expanding your network so that it is fit for the future. MileGate combines carrier grade broadband access, telephony and data interface in one single, compact access platform.

By using MileGate you can migrate whole or parts of your telecommunications network to the NGN. Expand your range of services to include new, high quality Triple Play and broadband business services, and continue to provide the range of traditional telephony and data services at the same time, without having to rely on two systems.[1]

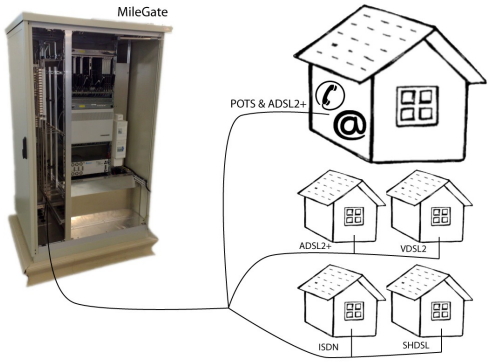


Illustration 6: MileGate

The system has one core unit and the possibility to plug 20 other units with different interfaces. As an example the MileGate provides up to 960 xDSL or 456 COMBO connections (POTS and ADSL2plus).

1.3.1 Flexibility in interfaces

Wide range of interfaces that can be mixed:

- POTS (Plain Old Telephone Service)
- ISDN
- ADSL/ADSL2/ADSL2plus
- VDSL2
- SHDSL
- COMBO solution (POTS and ADSL2plus)
- Optical Ethernet (100BaseFx or GbE)
- Electrical Ethernet (100BaseT)
- Legacy data interfaces (E1, V.35, V.36, X.21)

[1] http://www.keymile.com/media/en/internet/products/milegate/z_brochures/MileGate_Product_Overview.pdf

1.4 General structure of the MileGate

This section describes the important parts of the MileGate structure and some mechanisms needed for the implementation of the interfaces. This section provides an abridged version of the annexe:HTML Service.

1.4.1 Structure of the Object Model

The structure has been studied on the basis of the actual MileGate Configuration Software Tool (MCST) and the document "Introduction to the MileGate XML Management Interface".
We have to clarify at the very beginning that no direct access to functions on the embedded system is provided. Information exchange with the MileGate needs to be modelled according to the Managed object model (MOM). The managed objects (MOs) are an abstract view of resources (i.e. physical or logical parts of the equipment to be managed).

The tree of Managed Objects (MO) builds a hierarchical model.

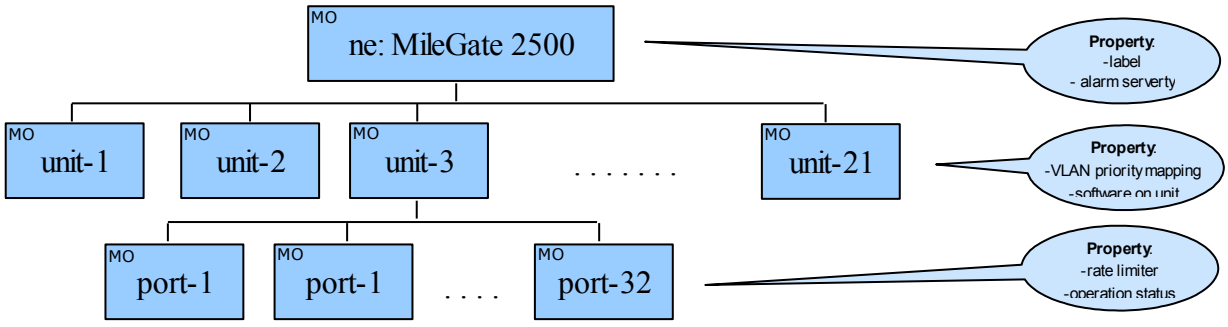


Illustration 7: MileGate Object Model structure

Each MO has its proper set of Management Functions (MF) depending on the type. The possible management functions are: Main, Configuration, Fault Management, Performance Management and Status.
This properties are the parameters that can be modified over the interface.

Additional information about the complete structure of the MileGate 2500 management functions, the mechanism for discovering the connected units, structure of the MileGate Accesspoint Description File (ADF), MCST GUI generation mechanism or MCST adaptation mechanism are represented as "annexe: HTML Service" on the chapter "3.3 Object Model and the actual management system MCST".

1.5 Constraints for MileGate

Due to the fact that MileGate is running embedded, there are some constraints we have to mention for the definition of our services.

1.5.1 Processor

The actual management system (MCST) generates a lot of request towards the management interface. Amelioration is possible but not vital.

Performance limitations rather have to be considered at the implementation of the HTML Service, HTTP Servers and the transformation Services. The generation of the HTML files and its storage uses much more system resources the the sending of KOAP messages .

CPU:	PowerPC 603E (~400MHz)

1.5.2 Memory

The memory of the MileGate is limited and has to be used with fully aware. The program itself need to be adapted to the KEYMILE coding rules.

The number of saved HTML pages should be as small as possible (We have to be aware of the huge number of properties for a fully equipped MileGate)

HTML is pure text and does not use lot of memory.

A memory footprint of a single program should not be bigger than 1 MB.

Core Card:	128MB / 256MB of RAM 128MB Flash Memory (no hard disk drive)

1.6 Communication with the MileGate

This section was important for the definition of the interfaces as they all have to communicate with the embedded system.

For the communication with the MileGate, each management interface has another manner to communicate. As example, the communication over USB has not lot of similarities with the communication over a command line client. An very important and for all the interfaces common part is the use of KEYMILE's proprietary KOAP messages. This communication is described briefly in this section.

1.6.1 Client-Server system

KOAP is a proprietary XML protocol which is transported over a proprietary message transport protocol.
It is a matter of a simple request-response system. The client is allowed to send request and the server (MileGate management interface) returns a response with the an indication whether the request was successful or had an error.
The KOAP protocol additionally offers the possibilities to send attachments.
All the services handling the configuration must access this management interface.

1.6.2 Format of the requests and responses

The following paragraph shows how the KOAP should look like.
The embedded system accepts request which looks as followed:

```
<?xml version="1.0" encoding="utf-8"?>
 <request version="1" seq="1" destAddr="/unit-1/port-1">
        <mdomain id="main">
              <operation seq="1" name="setLabel">
                    <Label>
                          <user>User1</user>
                          <service>Service1</service>
                          <description>Description1</description>
                    </Label>
              </operation>
        </mdomain>
 </request>
```

Code 8: KOAP request

The request addresses the Management Object Type (MO Type) "/unit-1/port-1" and the Management Function (MF) "main". The called function is named setLabel and requires the XML formatting shown .

To observe the response, we change from set to getLabel because the set function used just before won't deliver any content. The response looks as followed:

```
<?xml version="1.0" encoding="utf-8"?>
 <response version="1" seq="1" destAddr="/unit-1/port-1">
        <mdomain id="main">
               <operation seq="1" name="getLabel">
                      <execution status="success"/>
                      <Label>
                             <user>User1</user>
                             <service>Service1</service>
                             <description>Description1</description>
                      </Label>
               </operation>
        </mdomain>
 </request>
```

Code 9: KOAP response

The response has the same parameters as the request. Additionally the tag <execution> with the parameter status="success" has been added into the tag <operation>. An unsuccessful response would contain the execution parameter status="proc_error".
Within the <operation> tag, the values just send before in the setLabel function were returned.

2 Introduction into Web Services

Abstract

This chapter introduces the necessary knowledge concerning the Web Service and its architecture. The Service Oriented Architecture and the Web Service Architecture will be introduced.

2.1 SOA (Service-Oriented Architecture)

Before we introduce the Web Service Architecture, we need to mention some basics of the Service Oriented Architecture. This is necessary because the Web Service Architecture extends the Service Oriented Architecture.

W3C provides the following equation which interconnects the two words:

World Wide Web (WWW) + Service Oriented Architecture (SOA)
= Web Service Architecture

2.1.1 Architecture

The name indicates the basic idea behind this architecture, it is service oriented. We will not describe the SOA in detail, more information can be found under the references mentioned.

Main advantages of the SOA are that it facilitates manageable growth of enterprise systems and can reduce the costs for cooperation between organizations.

As most of the IT infrastructures and its organization have grown with a pillars-like architecture, the changeover to a SOA will be very difficult and time-consuming.

The following graphic illustrate this problem very well:

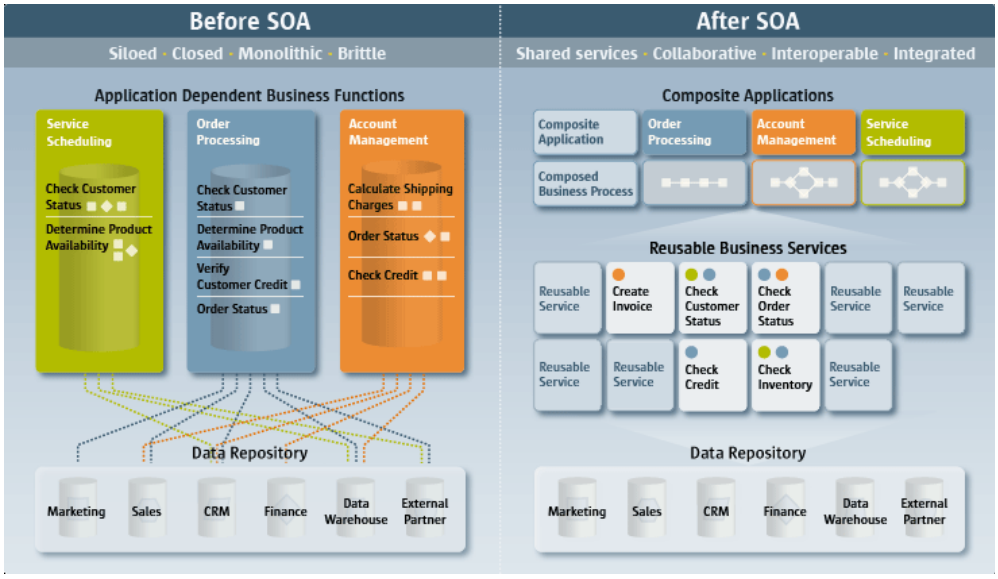


Illustration 10: Before & sfter SOA

The following illustration shows the famous triangle of Service Oriented Architectures. Roles are described briefly afterwards.

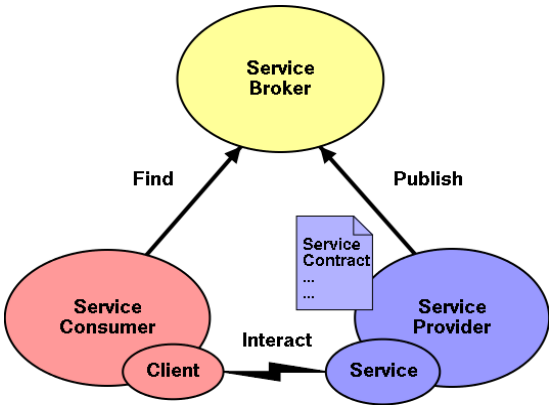


Illustration 11: Three roles in SOA

Service Provider

The service provider publishes the service. A description of the service is provided. The provider hosts and controls the access to the service.

Service Consumer

A service consumer interacts with the service via a service client. He can find services by querying the service broker. This role can be driven by an end user or by another service.

Service Broker (optional)

The service broker provides the directory service and allows service providers to publish and service costumer to find services. This role is optional, the service can also be found otherwise.

2.1.2 Basic characteristics of a SOA

A good summary of the basic characteristics of a SOA can be found in the technical library of IBM. The document is a recommendation to improve a Service Oriented Architecture and contains inter alia the following principles for a SOA.[2]

Guiding principles:

- *Reuse, granularity, modularity, composability*
- *Compliance to standards (both common and industry-specific)*
- *Services identification and categorization*

Specific architectural principles:

- *Separation of business logic from the underlying technology*
- *Single implementation and enterprise-view of components*
- *Life cycle management*
- *Efficient use of system resources*

[2] http://www.ibm.com/developerworks/webservices/library/ws-improvesoa/

2.2 Web Service Architecture

This chapter introduces the Web Service Architecture with its basic concept. We also want to introduce here the different organizations and task forces which standardize the concepts behind this architecture.
As we mentioned before, the Web Service Architecture extends a Service Oriented Architecture.

2.2.1 Definition

The definition of W3 published in the Web Services Architecture Requirements:[3]

« A Web service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols. »

2.2.2 Basic Concept

The basic components of a Web Service Architecture are:

- Communication
- Service Description
- Directory Service

The W3 recommends for the communication of Web Services the use of SOAP, its specification defines the XML-based message format and how it is embedded into a transport protocol. SOAP is mostly transported over HTTP but is not at all dependent on this transport protocol.
WSDL, also XML-based, is used to describe the Web Service.
Directory service specifies a standardized structure for the management of Web Service metadata. A possible directory service is UDDI. This service, which corresponds to the Service Broker of the SOA, is optional.

[3] Source: http://www.w3.org/TR/wsa-reqs/

2.2.3 Standardization

W3C[4]:

Founded in 1994 by Tim Bernes-Lee at the Massachusetts Institute of Technology, Laboratory for Computer Science (MIT/LCS) with support of the CERN in Geneva, the DARPA (Defense Advanced Research Project Agency) and the EU (European Union).

Multiple task forces are engaged in standards for HTML, XML, SOAP and WSDL. Interesting for the future will be standards as RDF (Resource Description Framework) and OWL (Web Ontology Language) concerning the semantic web.

OASIS[5]:

The Organization for the Advancement of Structured Information Standards, originally founded in 1993 as a cooperation of commercial enterprises, has its focus on standards of the topic e-business. Beside Web Services they provide techniques as UDDI, ebXML(electronic business using XML) and WS-BPEL(Business Process Execution Language).

IETF[6]:

The Internet Engineering Task Force defines more technique oriented standards and is therefore less conspicuous on Web Service design tasks. The most important standards by IETF are TLS (Transport Layer Security), LDAP (Lightweight Directory Access Protocol) and IPv6 (Internet Protocol version 6).

WS-I[7]:

Web Service Interoperability Organization does not publish any standards. The focus lies on the examination of concrete specifications and the implementation of different producers and guarantee the interoperability of them.

Profiles were defined to describe how to use the implementation of the different producers.

[4] http://www.w3.org

[5] http://www.oasis-open.org

[6] http://www.ietf.org

[7] http://www.ws-i.org

3 Traditional web service

Abstract

This chapter continues on the traditional web service introduced before and contains our project side issues “survey of embedded HTTP servers” and “service for HTML generation”.

3.1 Embedded http server

An embedded web server works the same way a standard web server does. The difference lies in its performance (usage, resources, memories, etc…) because it is downloaded on devices.
It allows to access, communicate and control remotely the device through a traditional browser or a particular interface of communication

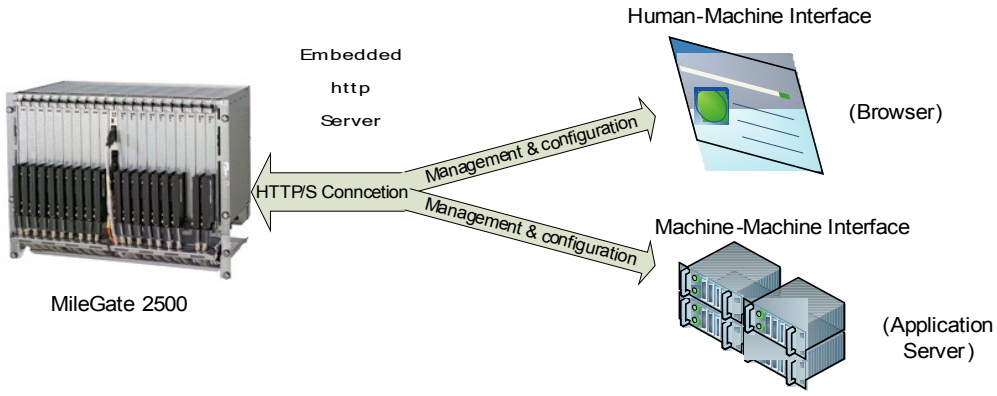


Illustration 12: Milegate case

For MileGate constraints (memory & processor limits) see 1.5.1 & 1.5.2

3.1.1 Selective criteria of servers

To compare different servers, we narrowed the long list of « light » servers. That list is presented in 2. We considered the following points to come to that conclusion:

Popularity: A popular application has an important FAQ and many forums that solve an array of problems.

Last release date: For security reason it is important to know how often the server is updated and the availability of updated and new versions.

Memory footprint: As the server will be used for embedded devices, it is very important to know the size and space the application needs to run as well as its effect on the processor.

Development language: According to KEYMILE' requirements, the development language can only be C or C++.

Operating System: The server will support at least an embedded Linux.

Multi-thread: use a separate process/thread for each request.

Poll: on sockets, process requests with a single process.

However, many of these servers are also available for Vxworks, Mac, FreeBSD, Posix and so on…

3.1.2 Required field

To properly compare different servers, we define three main domains of assessment. They are:

- Security
- Dynamic content
- Common functionality

3.1.2.1 Security

Three sub-categories exist:

Authentication: we evaluate the type of authentication that the server offers - Basic or Digest. The possibility to connect to an authenticated server such as RADIUS or KERBEROS is not considered because KEYMILE already fulfilled that requirement.

HTTPS: It is important to know if the server offers a secure connection (SSL/TLS protocol over HTTP) or not. In case it supports SSL/TLS, it is necessary to know if it is possible to use Mocana SSL Stack (now nanoSSL) to encrypt exchanged messages because KEYMILE already has bought a license of this software.
Remember that SSL/TLS authenticates endpoints and encrypts channels to provide session privacy and security on the internet.
Do not confuse IPSec which is an IP layer protocol (that enables the sending and receiving of cryptographically protected packets: TCP, UDP, ICMP…) and SSL/TSL which is an application layer protocol mostly utilized to protect HTTP transaction over TCP only.

OpenSSL: The possibility to use OpenSSL library so it remains free and more importantly OpenSource.

3.1.2.2 Dynamic content

This part informs on the technology to use to generate dynamically information that the server will make available. They are mainly:

CGI (Common Gateway Interface): with the help of a program written in any language (C/C++, Perl, Python…) it allows to generate information to be published through data provided by the user (client).

FastCGI: An Advanced form of CGI
Each http request is processed in a unique way with the standard GGI. Therefore, the server is overworked whenever there are important activities (many requests means many process). It is for that reason that FastCGI was built to process a bigger number of http requests.

SSI/SSJ (Server Side Includes / Server Side JavaScript)

- ***SSI*** is a programmable technical for web application. It allows helping and building an HTML document from several documents. In fact, it is a code generated out of the thin air and included in an HTML page when browsing other documents. The server later translates the HTML document and displays it to the user.
- ***SSJ*** it is simply a javaScript for a server. It is increasing in usage as it allows to generate a script by the server for the client. The script is sent and executed on the user's interface (browser), and then displayed information according to the results obtained on the user's device.

C/C++ (Dynamic page scripting in C/C++) : a C/C++ module is integrated with the server compiled binary code; without an intermediate layer as CGI or PHP and it is specially suited for low-resource (embedded) systems. This allows dynamic pages to be written in C/C++.

3.1.2.3 Common functionality

We tried to group other valuable data:

Compression: Possibility to compress the exchanged messages.

Segmentation or chunking: Useful when exchanged messages are bigger than 1500 Octets.

Session / cookies: Possibility to use sessions and manage cookies.

Documentation : if a fonctionnality description document is available.

Support IPv6: If IPv6 is supported

3.1.3 List of few embedded servers

On the table below, there is a list of several servers designed for different usage. There are web servers that can be embedded on devices.

Legend: " ✓ " means the feature is present otherwise "-"
☺☺ : very popular ; ☺ popular ; 😐 : little used ☹ : rarely used
?? means feature not found

Overview									
Server name	O.S	Dev. Language	Generate dynamically web pages	License / Open Source	Shareware / free	Multi-thread / Poll	Memory Footprint (KB)	Widely used	Release Date (Stable Version)
AppWeb	Embedded Linux…	C++	Yes	Dual License* / yes	✓ / ✓	✓ / ✓	800	☺☺	2008-03-14 (2.4.2)
GoAhead	Embedded Linux…	C	Yes	Dual License / yes	✓ / ✓	- / ✓	60	☺	2003-12-02 (2.1.8)
Klone	Unix, Linux, Windows…	C	Yes	Dual License */ yes	✓ / ✓	- / ✓	110→350	😐	2009-03-06 (2.2.0)
NicheStack HTTPServer	Any 16 or 32bit embedded	C	Yes	Closed source	✓ / -	- / ✓	??	☹	?? (3.1)
Barracuda Web Server	Embedded system (Linux…)	C	Yes	Dual License / No	✓ / -	✓ / ✓	**	☺	??

RomPager	Any Operating system	C	Yes	Commercial License / No	✓ / -	- / ✓	**	☺☺	??
Fusion http Server	Embedded devices	C	Yes	Commercial License / No	✓ / -	✓ / ✓	7→11	☹	??
Boa	unix	C, perl	Yes	GPL / Yes	- / ✓	??	120	☹	2005-02-23 (0.94.14)
Lighttpd	Unix, Linux, Windows…	C	Yes	BSD/ Yes	- / ✓	- / ✓	??	☺	2009-06-19 (1.4.23)
nginx	Unix, Linux, Windows…	C	Yes	BSD/ Yes	- / ✓	?? / ✓	480	☺	2009-06-22 (0.7.61)
cherokee	Unix, Linux, Windows…	C	Yes	GPL / Yes	- / ✓	?? / ✓	686	??	2009-07-01 (0.99.20)
Obelisk-Http	Any OS with Python	Python	Yes	GPL / Yes	- / ✓	✓ / ✓	50	☹	2007-05-28 (0.4.4)
WT	Unix, Linux, Windows…	C++	Yes	Dual License* / yes	✓ / ✓	✓ / ✓	250	☹😐	

Table 13: List of embedded servers

* = GPL (General Public License) and commercial licenses
**= The official web site writes "small footprint" but doesn't precise how small is it.

3.1.4 Comparative table of few servers

3.1.4.1 Free servers

Features											
	Security			Dynamic Content				Common functionality			
Server name	Authentification (Basic, digest..)	SSL/ which SSL stack	OpenSSL	CGI / Fcgi	PHP	SSI/SSJ	C /C++	Session / Cookies	Compression / chunking	IPV6	Doc
AppWeb	✓ / ✓	Yes/MatrixSSL & OpenSSL	Yes	✓ /-	Yes	✓ / ✓	Yes	✓ / ✓	- / ✓	Yes	Yes
Klone	✓ / ✓	Yes/OpenSSL	Yes	✓ /-	Yes	✓ / -	Yes	✓ / ✓	✓ / -	??	Yes
GoAhead	- / ✓	Yes/ ??	??	✓ /-	No	- / ✓	No	No	No	??	??
Lighttpd	✓ / ✓	Yes/OpenSSL	Yes	✓ / ✓	Yes	✓ / -	No	No	✓ / -	Yes	Yes
Nginx	✓ /-	Yes/OpenSSL	Yes	- / ✓	No*	✓ / -	No	No	✓ / ✓	Yes	Yes

Table 14: HTTP Free HTTP Servers

* : Not directly supported but Nginx supports FastCGI technology to work with many external tools and servers. PHP itself can be runned as FastCGI application and can process FastCGI requests from nginx.

3.1.4.2 Shareware servers

Features												
	Security			Dynamic Content						Common functionality		
Server name	Authentification (Basic, digest)	SSL& which SSL stack	OpenSSL	CGI / Fastcgi	WSDL compiler / SOAP API	PHP	SSI/SSJ	C /C++	Xml → C compiler	Compression / chunking	IPV6	Doc
RomPager [2]	??		??	??	✓ / ✓	??	??	Yes	Yes	??	??	??
NicheStack HTTPServer	✓ / ✓	Yes/NicheStack SSL	No	✓ / -	No	No	✓ / -	Yes	No	- / ✓	Yes	??
Fusion Web [2]	??	??	??	??	✓ / ✓	??	??	Yes	Yes	✓ / ✓	Yes	??
WT	No	Yes/OpenSSL	yes	- / ✓	??	No	- / ✓	Yes	??	✓ / ✓	??	??

Table 15: Shareware HTTP Servers

[2] :
Fusion Web Product Suite of Unicoi System. Unicoi's Web Product Suite includes the WebPilot Browser, Fusion HTTP Client & Server, Fusion XML SAX Parser, Fusion XML DOM Parser, Fusion XML-to-C Compiler, Fusion SOAP Client & Server, and Fusion WSDL Compiler.
See http://www.unicoi.com/product_suite_pages/fusion_web_product_suite.htm for more details...

RomPager is one of products of Allegro company. It offers a set of products which can be very interesting for usage (RomXML, RomSOAP, RomWebClient Secure…)
See http://www.allegrosoft.com/products.html for more details...

To get more information we must send a mail to those companies. What we have not done. Otherwise it would be interesting to know the true potential of their products.

3.1.5 Classification of compared servers

The following tables are used to classify the different servers, according to the needs of Keymile.

	C/C++	Support Nano SSL	OpenSSL	Compress / Chunking	Other SSL Stack	Auth : B /D	Free/Open Source (F/O)
credits	5	4	2	2/2	1	1/1	1/3

Table 16: HTTP server classification credits

Memory footprint	≤ 200 KB	≤ 400 KB	≤ 600 KB	≤ 800 KB	≤ 1 MB
credits	8	6	3	2	1

Table 17: HTTP server classification memory footprint

Servers	C/C++	Nano	Other ssl	OpenSSL	Com/chunk.	Auth B/D	Foot-print	F/O	Total
AppWeb	5	-	1	2	2	1+1=2	2	3	17
Klone	5	-	1	2	2	1+1=2	6	3	21
GoAhead	-	-	1	-	-	1	8	3	13
Lighttpd	-	-	1	2	2	1+1=2	1	1+3=4	12
Nginx	-	-	1	2	2+2=4	1	3	1+3=4	15
WT	5	-	1	2	2+2=4	-	6	3	21

Table 18: HTTP server classification

3.1.6 Few suggestions of server web generating content dynamically

3.1.6.1 Web server with incorporated module C/C++

According to our point system, Klone and WT SDK are the http servers that match the most with Keymile' requirements. So, for web server with incorporated module C/C++, it’s better to use

KLone is a fully-featured, multiplatform, web application development framework, targeted especially for embedded systems and appliances.
It is a self-contained solution which includes a web server and an SDK for creating web sites with both static and dynamic content.
When using KLone, there's absolutely no need for any additional component: neither the HTTP/S server (e.g. Apache, Netscape, Roxen), nor the typical active pages engine (PHP, Perl, ASP, Python).
KLone does everything, and does it fast and small.
KLone blends the HTTP/S server application together with its content and configuration into a single executable file.

The site developer writes his/her dynamic pages in C/C++ (in usual scripting style: <% *code* %>) and uses KLone to transform them into embeddable, compressed native code with the native C/C++ compiler.

It's also possible to use WT library.
In fact Wt is a C++ library and application server for developing and deploying web applications. It is not a 'framework', which enforces a way of programming, but a library.
A web application developed with Wt is written in only one compiled language (C++), from which the library generates the necessary HTML/XHTML, Javascript, CGI, SVG/VML/Canvas and AJAX code. The responsibility of writing secure and browser-portable web applications is handled by Wt.

However, it is possible to generate web pages with other processes.

3.1.6.2 Web server using SSI / SSJ

Another way to generate contents dynamically is the using of SSI or SSJ. We propose AppWeb for that technology. It possible to use Server Side JavaScript its JavaScript framework (Ejscript) as well as a Server Side Includes via CGI programming and its C++ library as another alternative for development.

The major disadvantage is the weight of the memory footprint: 800 KB- to heavy?!

3.1.6.3 Web server with incorporated module CGI / FastCGI

regardless of AppWeb, we can use some web servers which can also generate dynamic contents by using CGI or FastCGI programming. Lighttpd and Nginx are some examples.

3.2 HTML generation service

The objective of this section is according to the project objectives to find a good way to generate on the fly HTML pages within the MilGate which is providing a web browser access.
In this section, the necessary background will be provided, the feasibility studied and at the end a proposition for the implementation given.
This section provides an abridged version of the chapters 2, 4 and 5 of the "annexe:HTML Service".

3.2.1 Background

It would be interesting to offer a possibility to display and modify the configuration of the MileGate network device for humans. The most simple and standardized way is to provide the access via a web browser as a lot of other network devices as routers, modems, access points or switches do.
Our only interface to access the data or configuration parameters is the MileGate Object Model with its proprietary communication protocol.

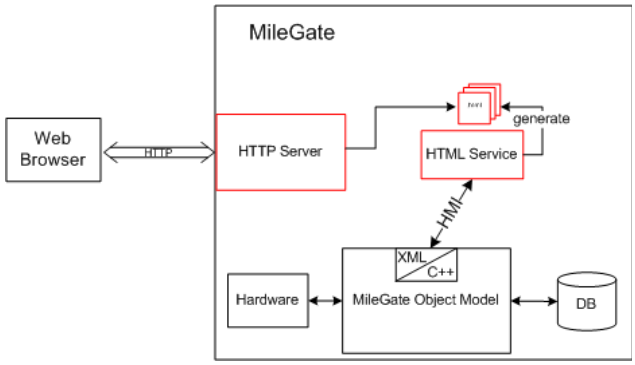


Illustration 19: System structure

For further treatment of the data for the presentation layer, we need to know the overall structure of the configuration (possible parameters) which needs to be parsed from an XML Schema, the ADF (proprietary AccessPoint Definition File) or in the future from the description of our Web Service (WSDL File).

3.2.2 Business functions

The aim is to analyse the feasibility of a service on the MileGate which creates HTML pages on the fly (run-time). It must be possible to change the configuration of the MileGate via an web browser.
It is not possible and not wished to to have the complete information in the memory because we would create redundancy which is complex to manage.

It is imaginable to save the navigation structure on the system but all the data will be requested on use.
The service must be adaptable with a modular structure. Also the presentation layer and the logic must be separated strictly.

3.2.3 Feasibility study

3.2.3.1 Identify problems for implementation

Problem	Description	Mitigation
Parsing HTML	It is difficult to extract information from HTML pages as it doesn't have a well defined structure.	The parsing of XML is much easier in C++. It could be a good solution to use XHTML instead of HTML.
Memory limitation for complete database. We can either create a DB for the service or always request the wanted parameters.	DB: + must get just modified parameter for regeneration - memory Direct output: + simpler to implementation - content of entire page must be requested on each modification	Creation of a DB probably won't be necessary for this implementation. We create additional problems caused by duplicating the data. Likewise is the implemented SAX parser on MileGate not optimal for the creation of a DB. It's better to keep the number of request as small as possible.
Menu structure	The menu is complex but needs to be well arranged at the same time.	A good technique to use would be a solution based a tree menu (example JavaScript) for the navigation within the nodes and kind of pop-up menu for the management functions.
Refresh of navigation menu on insertion of new unit	The menu must be updated (rewrite HTML page) if a new unit is inserted. On the browser it can be reloaded automatically with a refresh timing.	We need to detect the low-level interrupt! To find the accurate method the survey of the MCST will be helpful.
File transfer on HTML	The actual system initiates file transfer with a tag and adds the file just behind. This is possible due to the protocol is no standardized.	Here we have to study how to use HTTP/Put in C++
Acknowledge on modification	If the user modifies a parameter, he needs to be sure	We can not send messages to the user with HTML (HTTP Server is

	that the operation was successful.	between service and client). The only possibility is to print error messages on the HTML page which will be visible on the next reload.
Config of multiple Managed Objects (MO's)	The MCST GUI offers the possibility of configuring multiple MO's with one action.	This is difficult to implement in HTML, the task needs further studies.
Connection Manager (access the node)	The MCST GUI offers a connection manager which is user dependent.	The connection parameters of the users can not be managed trough the server. It is possible to use cookies to save connection parameters on the users web browser.
Customizing the GUI / Custom toolbar	A helpful add-on of the MCST is the customizable interface.	It will be very challenging to implement a customizable HTML page. The feasibility and its advantages should be studied in a further task. A custom toolbar is rather conceivable. It must also be saved on the client machine with a technologie such as cookies.
Printing option / Table CSV export	The MCST GUI offers a printing option and table export possibilities for spreadsheet programs.	Printing in HTML is obtainable with a well formated page or a additional stylesheet. The export possibility is more difficult and probably not supported in HTML. The CSV files may need to be generated within our HTML Service instead.

Table 20: Problems of HTML service and mitigation

3.2.4 Recommendation for Implementation

This topic contains our recommendation for the implementation of the HTML Service and an example user interface. The recommendations are based on the prior studies and converge in the basic structure towards the actual management system. This was necessary because no deep study on the structure of the look and feel was performed and with this, no change can be recommended.

We want a product which is as modular and adaptable as possible. To achieve this we certainly need a strict separation between logic and presentation.

3.2.4.1 Operation of the HTML Service

At the initiation of the HTML service, the entire navigation structure has to be generated. The result of this will be accessible by the client after the step 7 of the Sequence Diagram. The connection itself does not evoke the initiation of the service, the structure needs to existing already at this point of time.

The following points visualize the basic functionality of the service and describe how the service can figure out the structure of the node.

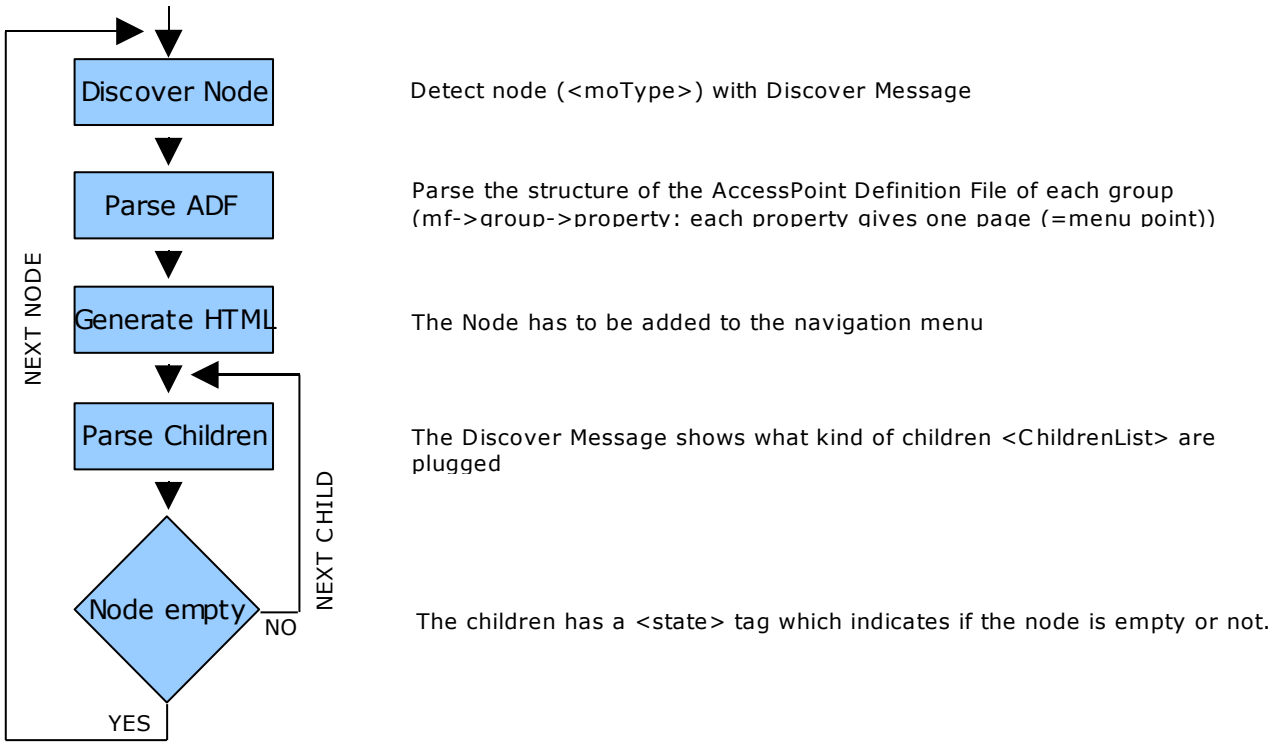


Illustration 21: Operation of the HTML Service

It has to be said that the parsing of objects has to be recursive which is not represented in this flowchart.

3.2.4.2 GUI Prototype

The menu is the most important part of the website because it defines the way we can navigate trough the sites and with this the ease of use. Basically we have the root node with its units and ports. Further a technique need to be evaluated to add maximal five additional menus to access the further navigation structure (Main, Configuration Management, Fault Management, Perform-

ance Management and Status) of each node. Possibilities are a second navigation frame or a pop-up accessible by the right mouse button.
The following illustration provides a GUI prototype with a second navigation frame and pull down menus.

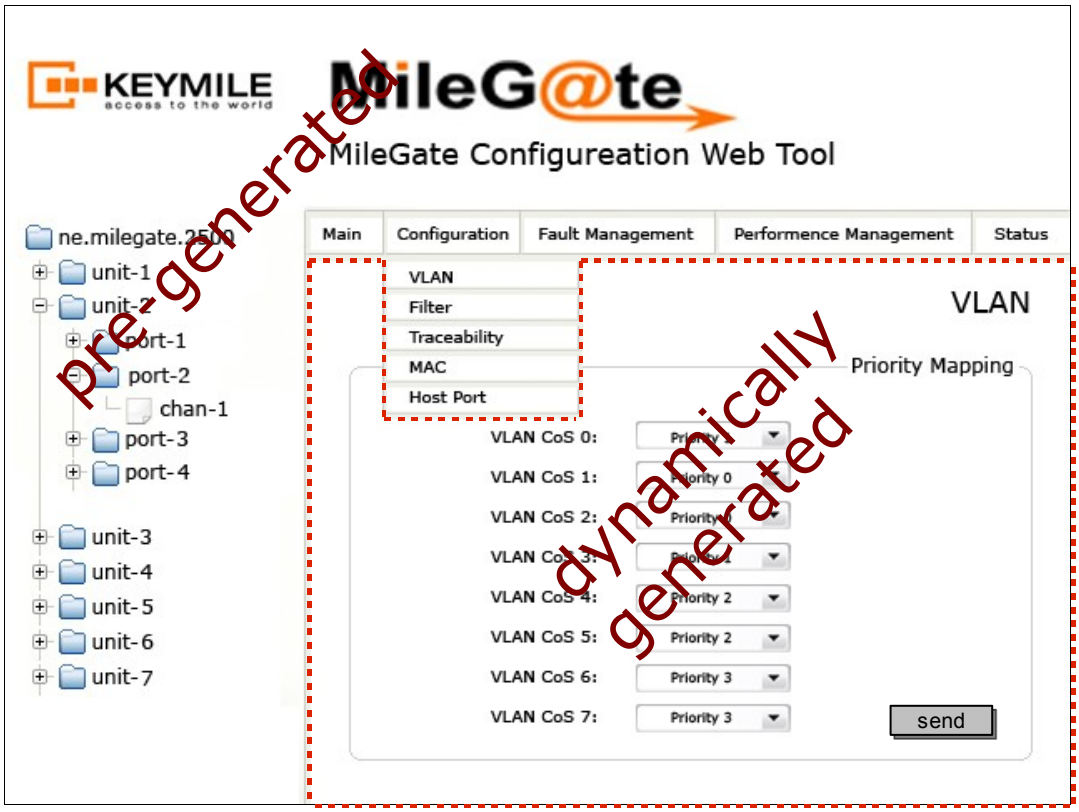


Illustration 22: GUI prototype

The website needs to be built with frames. That way we can use one single menu (left) on every other page. In function of the selection on the left side, the top menu and its menu points (top) need to change. As described before, the structure of this menu is defined in the ADF file for any possible kind of node.
If the user clicks on a navigation point, the real task for the service has to be performed. As we do not want to save the pages with the parameters anywere, we have to generate the entire content (exclusive of menus) at this point of time.

Request of content:

This list describes the activities of the program on a request of any content elements.

1. The possible form fields, check boxes, combo boxes, tables or buttons of one content frame are defined in the ADF file and need to be parsed.

2. Transformation between ADF XML and HTML/XHTML has to be performed
3. To get the values we have to send KOAP messages with indication which parameters we would like (in the example case it would be: request destAddr="/", mdomain id="cfgm", operation name="getPriorityMapping")
4. The service needs to merge the XHTML code and the parameters
5. Finally the XHTML has to be saved on memory
6. With a proper configuration of the HTTP Server, the file is now accessible by the user

3.2.4.3 Reaction on changes

The system needs to react to modification automatically. Modifications are possible on different interfaces such as CLI, MCST, syslog, SNMP and of course the web interface for this service.

The MileGate generates notification on a change of the configuration. Those notifications need to be captured by our service and as a consequence, the new navigation must be generated. This needs to be considered at the conception of the navigation structure.

If a new unit is added or removed, the node needs to be added in the navigation menu and of course also deleted. In this case, a similar mechanism as the one described in the under "Operation of the HTML service" has to be performed starting at the added unit instead of the root node.

3.2.4.4 Problems

Additional to the identification of the problems in the point "Feasibility studies" we list here some very important points for the implementation of the service.

Error Handling:

To announce errors to the user, we can just use the output of the HTML page. It is possible to generate error pages or to add the error message at any place of the page. We need to define what will be shown during the generation process to give the best feedback to the user.

Concurrent Problems:

The handling of concurrent access need to be checked to guarantee the functionality. In some cases, the access or the files have to be locked for secondary users.

Refresh Problems:

Automatic refresh of the HTML page with a refresh delay could cause some problems. We also have to pay attention that the caching mechanism of the browser/website is configured well. We need a very quick refresh time to not confuse/irritate the user.

Performance Problems:

We saw in the analysis that the embedded system has some limitations such as the performance. To avoid performance problems, proper testing is necessary.

3.2.5 Conclusion

A service which generates HTML pages on the MileGate is feasible.

The aim (advantage compared to MCST) and the wanted functions of such a service need to be planed and analysed carefully. Main advantage is that a client does not need to install anything. It is also imaginable that browsers on mobile devices can be used for configuration or supervision.

At our point of view, a customizable user interface or a change of the look-and-feel could bring some advantages for the use of the interface.

The required time to realize this project is very difficult to estimate at the actual state because it depends heavily on the desired functionalities.

The survey of the actual management tool (MCST) and its implementation helped a lot and completed the introduction into the very complex MileGate system. We feel certain that this analysis will facilitate future tasks and helps if such a service will be designed.

4 W3C Web Service Description

Abstract

This chapter introduces the Web Service Description Language WSDL and its structure. The structure is showed by means of abstracts of the final Web Service Description for the MileGate interface.

For a better understanding, the link between the WSDL and the exchanged SOAP message has been added.

The complete description of the Web Service (WSDL file) can be found as annexe:WSDL. The modified parts are highlighted and resumed in the chapter “7.3 WSDL File generation”.

4.1 Introduction

WSDL is an XML language for describing Web Service interfaces. The language is standardized by the W3C.
The specification of the version 1.1 exists since 2001. The follower version (2.0) reached the status of a 'W3C Recommendation' in march 2006 but most of the current Web Services still use the previous version.

The description of such a service is spit into two parties, we have an abstract and a concrete description. The abstract view focuses the functionality and the concrete enters more into the technical detail. Thus we have a separation between the details and the manner our service is offered.

The components of the description are:

Abstract:	Operation
	Messages Exchange Pattern
	Interface
Concrete:	Binding
	Endpoint
	Service

The main difference of WSDL according to other description languages for interfaces (e.g. IDL, Interface Description Language) is that everything is concentrated in one file. We are able to communicate with the service just on the base of the WSDL file. Of course we have also the possibility to write the description modular (include, import) to provide better legibility and maintainability.

4.2 Structure of the description

We want to introduce briefly the elements used to describe the Web Service and show afterwards a few more details using the description of our interface. If two elements are used to describe one single element, this is due to the different versions of WSDL. The first element belongs to the version 1.1 and the second to the standard 2.0.

definitions / description (root element)
This XML element represents the root element of the WSDL file and defines the different name spaces.

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="mob_mainbase"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:mob_mainbase_xml="http://keymile.com/milegate/ws/mob_mainbase_xml"
   xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap12/"
   xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/06/management"
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
   targetNamespace="http://www.keymile.com/milegate/ws/mob_mainbase_xml">
```

Code 23: WSDL definitions

documentation
The section documentation contains a textual annotation to the service.

```
   <documentation>
      -textual description of Web Service
      -further infos for the use of this service or interface
      -contact person
   </documentation>
```

Code 24: WSDL documentation

types
Defines the usable data types.

```
<types>
    <xs:schema
    xmlns="http://www.keymile.com/milegate/ws/mob_mainbase_xml"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:_0="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    targetNamespace="http://www.keymile.com/milegate/ws/mob_mainbase_xml">
       ...
       <xs:element name="Label" type="Label__Type"/>
         <xs:complexType name="Label__Type">
            <xs:sequence>
```

```
            <xs:element name="user">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="63"/>
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="service">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="63"/>
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="description">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="127"/>
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
      <xs:element name="severity" type="severity__Type"/>
      <xs:simpleType name="severity__Type">
         <xs:restriction base="xs:string">
            <xs:enumeration value="notification"/>
            <xs:enumeration value="cleared"/>
            <xs:enumeration value="indeterminate"/>
            <xs:enumeration value="warning"/>
            <xs:enumeration value="minor"/>
            <xs:enumeration value="major"/>
            <xs:enumeration value="critical"/>
         </xs:restriction>
      </xs:simpleType>
      ...
```

XMLSchema base data types

Code 25: WSDL types

message

This element contains the possible messages and the types which are allowed to use.

```
<!-- WS-Management headers  -->
 <message name="ResourceURIMessage">
    <part name="Header" element="wsman:ResourceURI"/>
 </message>
 <message name="SelectorSetMessage">
    <part name="Header"  element="wsman:SelectorSet"/>
 </message>

 <!-- WS-Addressing headers  -->
 <message name="ToMessage">
```

```
      <part name="Header"  element="wsa:To"/>
   </message>
   <message name="ReplyToMessage">
      <part name="Header"  element="wsa:ReplyTo"/>
   </message>
   <message name="ActionMessage">
      <part name="Header"  element="wsa:Action"/>
   </message>
   <message name="MessageIDMessage">
      <part name="Header"  element="wsa:MessageID"/>
   </message>

   <!-- bodys  -->
   <!-- FAULT MESSAGE -->
   <message name="errorMessage">
      <part name="Error" element="mob_mainbase_xml:Fault"/>
   </message>
   ...
   <message name="Discover__Message">
      <part name="Body" element="mob_mainbase_xml:Discover"/>
   </message>
   ...
      <message name="Label__Message">
      <part name="Body" element="mob_mainbase_xml:Label"/>
   </message>
   ...
```

Reference to data type (TYPES)

Code 26: WSDL message

port type / interface

Describes the interfaces and the provided operations on this interface. For each operation the corresponding input and output messages are listed.

```
   <portType name="main_base__PortType">
      <!-- MAINBASE PORT TYPES -->
      <operation name="GetLabel__Operation">
         <input message="mob_mainbase_xml:EmptyMessage"/>
         <output message="mob_mainbase_xml:Label__Message"/>
         <fault name="Fault" message="mob_mainbase_xml:errorMessage"/>
      </operation>
      <operation name="SetLabel__Operation">
         <input message="mob_mainbase_xml:Label__Message"/>
         <output message="mob_mainbase_xml:Label__Message"/>
         <fault name="Fault" message="mob_mainbase_xml:errorMessage"/>
      </operation>
      <operation name="GetAlarmSeverity__Operation">
         <input message="mob_mainbase_xml:EmptyMessage"/>
         <output message="mob_mainbase_xml:AlarmSeverity__Message"/>
         <fault name="Fault" message="mob_mainbase_xml:errorMessage"/>
      </operation>
      ...
      <operation name="GetDiscover__Operation">
```

The input message (view of Service) doesn't have any body!

```
        <input message="mob_mainbase_xml:EmptyMessage"/>
        <output message="mob_mainbase_xml:Discover__Message"/>
        <fault name="Fault" message="mob_mainbase_xml:errorMessage"/>
      </operation>
      ...
```

Code 27: WSDL portType

binding

With the element binding we declare which transport protocol is used for which interface. For inputs or outputs of operations we need to assign the messages to the elements of the transport protocol (for the example SOAP, this will be SOAP:body or SOAP:header)


```
   <binding name="main_base__Interface"
        type="mob_mainbase_xml:main_base__PortType">
      <soapbind:binding style="document"
       transport="http://schemas.xmlsoap.org/soap/http"/>
      <!-- MAINBASE BINDING -->
      <operation name="GetLabel__Operation">
         <soapbind:operation
        soapAction="http://schemas.xmlsoap.org/ws/2004/09/transfer/Get"/>
         <input>
            <soapbind:header message="mob_mainbase_xml:ResourceURIMessage"
              part="Header" use="literal" />
            <soapbind:header message="mob_mainbase_xml:SelectorSetMessage"
              part="Header" use="literal" />
            <soapbind:header message="mob_mainbase_xml:ToMessage" part="Head
              er" use="literal" />
            <soapbind:header message="mob_mainbase_xml:ReplyToMessage"
              part="Header" use="literal" />
            <soapbind:header message="mob_mainbase_xml:ActionMessage"
              part="Header" use="literal" />
            <soapbind:header message="mob_mainbase_xml:MessageIDMessage"
              part="Header" use="literal" />
            <soapbind:body use="literal"/>
         </input>
         <output>
            <soapbind:header message="mob_mainbase_xml:ResourceURIMessage"
              part="Header" use="literal">
            </soapbind:header>
            <soapbind:header message="mob_mainbase_xml:SelectorSetMessage"
              part="Header" use="literal" />
            <soapbind:header message="mob_mainbase_xml:ToMessage" part="Head
              er" use="literal" />
            <soapbind:header message="mob_mainbase_xml:ReplyToMessage"
              part="Header" use="literal" />
            <soapbind:header message="mob_mainbase_xml:ActionMessage"
              part="Header" use="literal" />
            <soapbind:header message="MessageIDMessage" part="Header"
              use="literal" />
            <soapbind:body use="literal"/>
         </output>
<fault name="Fault">
   <soapbind:fault use="literal" name="Fault" />
</fault>
```

```
      </operation>
      <operation name="SetLabel__Operation">
         <soapbind:operation
        soapAction="http://schemas.xmlsoap.org/ws/2004/09/transfer/Set"/>
         <input>
            <soapbind:header message="mob_mainbase_xml:ResourceURIMessage"
              part="Header" use="literal" />
            <soapbind:header message="mob_mainbase_xml:SelectorSetMessage"
              part="Header" use="literal" />
            <soapbind:header message="mob_mainbase_xml:ToMessage" part="Head
              er" use="literal" />
            <soapbind:header message="mob_mainbase_xml:ReplyToMessage"
              part="Header" use="literal" />
            <soapbind:header message="mob_mainbase_xml:ActionMessage"
              part="Header" use="literal" />
            <soapbind:header message="mob_mainbase_xml:MessageIDMessage"
              part="Header" use="literal" />
            <soapbind:body use="literal"/>
         </input>
         <output>
            <soapbind:body use="literal"/>
         </output>
         <fault name="Fault">
            <soapbind:fault use="literal" name="Fault" />
         </fault>
      </operation>
```

No need for interpretation.Passes to application as a full XML

Code 28: WSDL binding

service

Describes where the service is located. 'Services' can be subdivided into 'port/endpoint' with different addressing parameters. See next paragraph for description of parameters.

```
   <service name="SetLabelService">
      <port name="SetLabelPort"
        binding="mob_mainbase_xml:main_base__Interface">
         <soapbind:address location="http://localhost:9357/wsman/"/>
         <wsa:EndpointReference name="labelEPR"
            xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl">
            <wsa:Address>http://localhost:9357/wsman/</wsa:Address>
            <wsa:ReferenceParameters>
               <wsman:SelectorSet>
                  <wsman:Selector name="mf">main</wsman:Selector>
                  <wsman:Selector name="property">/Label</wsman:Selector>
               </wsman:SelectorSet>
               <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
            </wsa:ReferenceParameters>
         </wsa:EndpointReference>
```

Definition of Service Endpointwith the addressing parameters

Code 29: WSDL service

4.3 SOAP Message

The SOAP message defined in the Web Service Description File helps a lot to understand the description.
We are going to represent the SOAP messages for the Set- and GetLabel operation.

GetLabel SOAP message:

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml"
    xmlns:wsa  ="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
    <soapenv:Header>
        <wsa:To>http://localhost:9357/man</wsa:To>
        <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
        <wsa:ReplyTo>
            <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/
              role/anonymous</wsa:Address>
        </wsa:ReplyTo>
        <wsman:SelectorSet>
            <wsman:Selector Name="mf">main</wsman:Selector>
            <wsman:Selector Name="property">/Label</wsman:Selector>
        </wsman:SelectorSet>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
        </wsa:Action>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued
        </wsa:MessageID>
    </soapenv:Header>
    <soapenv:Body>
    </soapenv:Body>
</soapenv:Envelope>
```

Addresse

Ressource

Property

Action: Get

Code 30: SOAP GetLabel

The input type (view of service) of the GetLabel message (defined in PortTypes):

```
<input message="mgws:EmptyMessage"/>     <!-- NO BODY -->
```

SetLabel SOAP message:

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml"
    xmlns:wsa  ="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
    <soapenv:Header>
        <wsa:To>http://localhost:9357/man</wsa:To>
```

```
        <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
        <wsa:ReplyTo>
            <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/
              role/anonymous</wsa:Address>
        </wsa:ReplyTo>
        <wsman:SelectorSet>
            <wsman:Selector Name="mf">main</wsman:Selector>
            <wsman:Selector Name="property">/Label</wsman:Selector>
        </wsman:SelectorSet>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
        </wsa:Action>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued
        </wsa:MessageID>
    </soapenv:Header>
    <soapenv:Body>
        <mob:Label>
            <mob:user>a</mob:user>
            <mob:service>b</mob:service>
            <mob:description>c</mob:description>
        </mob:Label>
    </soapenv:Body>
</soapenv:Envelope>
```

Action: Put

The SetLabel has a body of the typ 'Label__Type' as input message

Code 31: SOAP SetLabel

The input type (view of service) of the SetLabel message (defined in Port-Types):

```
<input message="mob_mainbase_xml:Label__Message"/>
```

5 Web Service Concepts

Abstract

There is a huge variety of concepts and standards for Web Services. Concepts are provided by the World Wide Web Consortium W3C[8], OASIS[9], Microsoft[10], IBM[11] and even more. Some of this concepts overlap.

This chapter discusses the used concepts for our Web Service and provides an selection of other concepts which has been defined during this project as interesting for the future development of the MileGate Web Service.

Most of these concepts had not been included in the actual Web Service Description (WSDL) for reasons of time constraints. For these concepts interesting points for KEYMILE are emphasized and commented.

[8] http://www.w3.org/2002/ws/

[9] http://www.oasis-open.org/specs/

[10] http://msdn.microsoft.com/en-us/library/ms951274.aspx

[11] http://www.ibm.com/developerworks/webservices/standards/

5.1 Addressing

The W3C recommendation Web Service Addressing 1.0 – Core[12] of the 9 May 2006 defines the construct of the message addressing properties and the end-point references.

Other recommendation describes the Web Service Addressing 1.0 – SOAP Binding[13] (9 May 2006), the Web Service Addressing 1.0 – Metadata[14] (4 September 2007) and the candidate recommendation Web Service Addressing 1.0 – WSDL Binding[15] (29 May 2006).

5.1.1 WS-Addressing

This recommendation provides a mechanisms for end-to-end addressing of messages independent of the transport protocol used.

Addressing properties are, with the use of SOAP, contained in the header block.

The use of WS-Addressing allows us to address the source and destination endpoint of the system and to provide a identity for the message. Additional we specifies an action URI which defines the expected semantics.

With the concept of SOAP binding we assign the exchange structure defined by SOAP and a set of predefined faults.

The WSDL Metadata and WSDL binding indicate if the service is using WS-Addressing and provides the possibility for different message exchange patterns such as one-way, request-response, notification and solicit-response for WSDL 1.1 and some more for WSDL 2.0.

5.1.1.1 Endpoint Reference EPR

Endpoint Reference is a concept introduced by WS-Addressing and is used for the dynamic generation and customization of service endpoints.

As we have in our system endpoints that can change with the modification of the configuration or with the insertion of new hardware, we need a mechanism to indicate the new endpoint.

[12] http://www.w3.org/TR/ws-addr-core/

[13] http://www.w3.org/TR/ws-addr-soap/

[14] http://www.w3.org/TR/ws-addr-metadata/

[15] http://www.w3.org/TR/ws-addr-wsdl/

Possibilities are an additional Web Service (Endpoint Manager) which provides information about the addressable endpoints. Such a Web Service is described on the apache website (http://svn.apache.org/repos/asf/cxf/trunk/testutils/src/main/resources/wsdl/locator.wsdl).
Other approaches are described later in the chapter 'WS-Distributed Management' under 'Advertisement' and 'Discovery'.

5.1.2 WS-Management

The final specification WS-Management was published by the Distributed Management Task Force DMTF the 02 December 2008. It provides a common way for systems to access and exchange management information.

The default addressing model uses a representation of an EPR that is a tuple of the following SOAP headers:[16]

- ***wsa:To*** *(required): the transport address of the service*
- ***wsman:ResourceURI*** *(required if the default addressing model is used): the URI of the resource class representation or instance representation*
- ***wsman:SelectorSet*** *(optional): identifies or "selects" the resource instance to be accessed if more than one instance of a resource class exists*

The ResourceURI is in our case used to address the Managed Object (e.g. /unit-11) and the SelectorSet specifies the management function (mf, e.g. Main) and the property (e.g. Label).

[16] http://www.dmtf.org/standards/published_documents/DSP0226_1.0.0.pdf (5.1.2 Default Addressing Model)

5.1.3 WS-Transfer

WS-Management has the status of W3C Member Submission (27 September 2006). The latest working draft is dated the 25 June 2009.

We use just the defined resource operations (which provides the synchronous message exchange we need) such as get and put with the URI:

```
http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
```

REMARK: In the latest working draft the URI changed to:

```
http://www.w3.org/2009/06/ws-tra/Get
http://www.w3.org/2009/06/ws-tra/Put
```

Additionally the resource operations delete and create are possible.

5.2 Resource

The concepts in this chapter describe the handling of resources with Web Services. Following specifications are published by OASIS, please pay attention on the status of the recommendation which is indicated at the beginning of each description.

5.2.1 WS-Discovery

WS-Discovery is not standardized yet and has the state of an OASIS Committee Specification 01 since 14 May 2009.[17]

It defines a discovery protocol to locate services. It is often used to discover structures like LDAP (Lightweight Directory Access Protocol) or similar directories.

As our system contains one single service per MileGate, we have no need of a discovery at this level. Discovery could be used to figure out the complete infrastructure (ensemble of MileGates). Actually, this function is not needed because the system architecture and its addressing is designed in advance and won't change over the time.

CAN'T BE USED TO DISCOVER THE MANAGED OBJECTS (RESOURCE) OF THE MILEGATE!

5.2.2 WS-Resource

WS-Resource became a OASIS Standard the 1 April 2006.

The goal of WS-Resource is to standardize the terminology and concepts needed to express the relationship between Web services and resources.[18]

[17] http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.pdf

[18] http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf (1.1 Goals and Requirements)

A resource is represented by an endpoint reference (EPR) and addressed using the WS-Addressing concept:

```
<wsa:EndpointReference>
  <wsa:Address>http://192.168.0.1?res=RessourceName</wsa:Address>
  …
</wsa:EndpointReference>
```

Code 32: WS-Ressource

The SOAP binding would look as followed:

```
<wsa:To>http://192.168.0.1?res=RessourceName</wsa:To>
```

Code 33: WS-Ressource SOAP binding

5.2.2.1 WS-Resource Properties[19]

WS-Resource Properties also became a OASIS Standard the 1 April 2006.

The goal of WS-ResourceProperties is to standardize the terminology, concepts, operations, WSDL and XML needed to express the resource properties projection, its association with the Web service interface, and the messages defining the query and update capability against the properties of a WS-Resource.

Resource Property:
A resource property is a piece of information defined as part of the state model of a WS-Resource.

Resource Properties Document:
The XML document representing a logical composition of resource property elements. The resource properties document defines a particular view or projection of the state data implemented by the WS-Resource.

5.2.2.2 Comment

This concepts offer another manner for addressing the MILEGATE property (e.g. Label) and its parameters (e.g. Label1).

- With GetMultipleResourceProperties we can get a selection of Resource Properties. This mechanism offers the possibility of a customized request

[19] http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf (1.1 and 2)

according to the preferences of the client. The advantage is that we do not have to request multiple operations and filter the content afterwards.

- With QueryResourceProperties we are able to query a Resource Properties document of a WS-Resource using a query expression such as XPath.
- The manageability of the system could be improved due to the dynamic add/delete of Resource Properties into the Resource Property document. (InsertResourceProperties, UpdateResourceProperties, DeleteResource-Properties)

DOES NOT HELP TO FIGURE OUT WHICH ENDPOINT IS SUPPORTED BY WHICH OPERATION!

5.2.3 WS-Notification

WS-Notification contains the standard WS-Base Notification, WS-Brokered Notification and WS-Topics.

5.2.3.1 WS-Base Notification

WS-Base Notification became a OASIS Standard the 1 October 2006.

The goal of WS-BaseNotification is to standardize the terminology, concepts, operations, WSDL and XML needed to express the basic roles involved in Web services publish and subscribe for notification message exchange.[20]

A notify message containing one or more notifications should look as followed:[21]

```
…
<wsnt:Notify>
  <wsnt:NotificationMessage>
    <wsnt:SubscriptionReference>
      wsa:EndpointReferenceType
    </wsnt:SubscriptionReference> ?
    <wsnt:Topic Dialect="xsd:anyURI">
      {any} ?
    </wsnt:Topic>?
    <wsnt:ProducerReference>
      wsa:EndpointReferenceType
```

[20] http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf (1.1 Goals and Requirements)

[21] http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf (3.2 Notify)

```
    </wsnt:ProducerReference> ?
    <wsnt:Message>
      {any}
    </wsnt:Message>
  </wsnt:NotificationMessage> +
    {any} *
</wsnt:Notify>
…
```

Code 34: WS-Base Notification

The notify message just before is transported as content of the SOAP body. Addressing for the notification (in SOAP header) by definition is following WS-Addressing action.

```
<wsa:Action>
  http://docs.oasis-open.org/wsn/bw-2/NotificationConsumer/Notify
</wsa:Action>
```

Code 35: Notification action

The concept for the management of the subscription is also defined in WS-Base Notification.

5.2.3.2 WS-Brokered Notification

WS-Topics became a OASIS Standard the 1 October 2006.

The goal of WS-BrokeredNotification is to standardize message exchanges involved in Web services publish and subscribe of a message broker.[22]

5.2.3.3 WS-Topics

WS-Topics became a OASIS Standard the 1 October 2006.

The goal of the WS-Topics specification is to define a mechanism to organize and categorize items of interest for subscription known as "topics". It defines a set of topic expression dialects that can be used as subscription expressions in subscribe request messages and other parts of the WS-Notification system.[23]

[22] http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf (1.1 Goals and Requirements)

[23] http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf (1.1 Goals and Requirements)

***Topic**:*
A Topic is the concept used to categorize Notifications and their related Notification schemas.

Topic Tree:
A hierarchical grouping of Topics.

5.2.3.4 Comment

The mechanism described in this standards is basically similar to the notification system used in the MileGate. The requirement for the notifications used for the logbook could be fulfilled with this technique without the need for a continuous polling. (Pull-style notifications also possible)
Information in the logbook subcategories alarm, configuration changes, session login, equipment changes and events can be made accessible in a more particular way for other purposes.
It is recommended to allow authorization policies for topics.

- The hierarchical structure of the topics allows a very targeted subscription for notifications.
- Management of the topics stays handy, also for large topic sets.
- The client can regroup the readout of notification according to his belongings and anywhere in his system.

5.3 Management

We already saw the WS-Management specification in the chapter Addressing. The idea behind this separation is that we just used WS-Management for addressing purposes.

In this chapter we describe functionality that goes much further. A complex concept is represented which interconnects multiple standards described before.

5.3.1 WS-Distributed Management

The standard WS-Distributed Management contains two parties.

Management using Web Services (MUWS 1.0) became a OASIS Standard the 9 March 2005 and Management of Web Services (MOWS 1.0) on 1 August 2006. We will discuss here just the first standard. The second standard (MOWS 1.0) will be more interesting for the implementation of the management interface and not for the definition of the interface.

5.3.1.1 Management Using Web Services

The following paragraph defines some necessary terminology defined in the MUWS specification.

Manageable resource:

A resource capable of supporting one or more standard manageability capabilities.

Capability:

A group of properties, operations, events and metadata, associated with identifiable semantics and information and exhibiting specific behaviors.

Manageability capability:

A capability associated with one or more management domains.

Manageability endpoint:

A Web service endpoint associated with and providing access to a manageable resource.

Management domain:
An area of knowledge relative to providing control over, and information about, the behavior, health, lifecycle, etc. of manageable resources.

Management Using Web Services (MUWS) enables management of distributed information technology (IT) resources using Web services. Many distributed IT resources use different management interfaces. By leveraging Web service technology, MUWS enables easier and more efficient management of IT resources.[24]

MUWS is based on number of other specifications such as WS-Addressing, Metadata, Endpoint Reference, WS-Notification, WS-Topics, WS-Discovery, WS-Resource Properties which have been introduced before.

Manageability capabilities

The following capabilities are summarized from the documents MUWS part 1[25] (Chapter 3) & 2[26] (Chapter 2 and 3) mentioned as reference. The capabilities describe how the service can be used.

Operations
The operations in the MUWS specification correspond to those used in WSDL (portType element containing operation element with a description and any relevant metadata).

Properties
The properties of a manageable resource use the mechanism defined in WS-Resource Properties and its resource properties document.

Events
Event types are defined by using 'topic' and 'message content' elements. The information in the second element is transmitted as a part of the notification message (defined by WS-Base Notification).
To support event classification, different SituationCategoryTypes (element) such as AvailabilitySituation, CapabilitySituation, ConfigurationSituation and so on were defined (full list on page 9 of MUWS part 2). The aim of this classification is that the event consumer can comprehend the situation according the ability of the event source.
For each capability, topics are defined to link the capability with the event.

[24] http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.pdf (1 Introduction)
[25] http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.pdf
[26] http://docs.oasis-open.org/wsdm/wsdm-muws2-1.1-spec-os-01.pdf

Metadata

We can define metadata on properties and operations. The aim of this is to provide information available in WSDL and WS-Resource Properties to a tool or management application.

With the metadata element 'ValidWhile', we are able to block the invocation of an operation if certain properties do not have certain values.

Operational Status

With the capability operational status we have can simply represent if a resource is 'Available', 'PartiallyAvailable', 'Unavailable' or 'Unknown'.

This function can be implemented using the notification on property value change provided by WS-Resource Properties.

Management-related capabilities

The function of a management-related capability is related to the management of a resource, but it is not necessarily offered directly by a manageability endpoint of a resource. For example, the capability to help a manageability consumer discover a new manageable resource can be provided by a registry instead of by a management representation of the resource. As another example, a manageable resource may provide information about relationships in which it participates.

The following capabilities are summarized from the documents MUWS 2 (Chapter 4 and 5) mentioned as reference.

Relationships

The relationship defines the association between resources and the role of each participant. Interesting point for the MileGate system is that we can define a common AccessEndpoint for the participants of a relationship. A relationship may have its own properties, operations, events, lifecycles or can provide information about the relationship.

Another good point is that with the definition of relationships we enable the discovery of Endpoint References for other resource that participates in the relationship.

Advertisement

This capability provides a mechanisms to notify the creation or destruction of manageable resources. The following four new event topics are defined by Advertisement:

- ManageabilityEndpointCreation
- ManageableResourceCreation
- ManageabilityEndpointDestruction
- ManageableResourceDestruction

On the creation of a new Endpoint, the most interesting case for the MileGate system, an associated 'CreationNotification' message (WS-Notification) delivers the new Endpoint Reference.

Discovery

The goal of discovery is to obtain the EPR of a manageability endpoint.[27]

The advertisement capability, just introduced before, provides one way to provide a discovery mechanisms via events.
Another possibility is the discovery mechanisms via relationships described in under 'Relationships'.

A last possibility, perhaps also interesting for the MileGate, is the discovery of manageable resource by invoking a query on a registry. It is recommended to use a registry of the type specified by the WS-Service Group[28] specification.
Therefore MileGate should provide such a registry.

5.3.1.2 Comment

This specification defines how the different concepts can be combined together and all the advantages from each of them can enhance the usability of the complete system. We have plenty of good mechanisms for the dissolving of the problems we get if we pass from a proprietary to a standardized solution using Web Services.

It follows a short recapitulation of the advantages.

Resource Properties

- customized requests
- query resource properties using Xpath
- better manageability on changes of the resource properties

Notification/Topic

- similarity to actual notification system
- hierarchical structure of topics
- subscription

[27] http://docs.oasis-open.org/wsdm/wsdm-muws2-1.1-spec-os-01.pdf (5 Discovery)

[28] http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf

Metadata

- constraints for the invocation of operations
- machine readable

Operational status

- knowledge if resource is available

Relationships

- common AccessEndpoint in relationship
- discovery of Endpoint References in relationship

Advertisement

- discovery of Endpoint References with creation notifications

6 Web Service Tools

Abstract

In this chapter we will discuss about interoperability between web services and the different validation tools recommended by the unavoidable organism.
We'll also talk about evaluated framework that can be used by the client to consume the described service

6.1 Clients Tools

There are few available platforms to create web applications. In the past, each application used its own specific protocol for service integration between devices. For that reason, applications that used different platforms could not communicate or shared data. This problem is called interoperability, which is the ability to communicate and share data effectively and efficiently. The awareness of those unfortunate limitations led to the standardization of data structure and shared data; thus, the study of web services observed in this project.

The objective of this project is to emphasize to the best of our (my) abilities the different aspects that could either prevent a good integration of web services or represent an obstacle to reach efficient software and traditional materials. In fact, the idea is to produce an interactive « Milegate » capable of understanding the service web standardized language that works with any client device regardless of the web tools used by the device.

6.1.1 Interoperability between web services and SOAP protocol

Service web technologies such as protocol SOAP, WSDL Language and HTTP protocol are currently used to transfer messages between machines. Those messages vary in complexity. It ranges from the type of methodology to the submission of an order. A common function – of higher level – of web service requires the implementation of RPC communication type (Remote Procedure Call, a long distance procedure call that allows a computer to run a program on another computer.

For the rest of the paper, we will focus on a list of practical and frequently asked questions concerning interoperability ; mainly questions that relate to RPC communication type using protocol SOAP.

It is important to remember that interoperability problems are not often linked to SOAP itself because it is « normalized »; on the contrary, it is linked to transport and core protocol XML used.

Therefore, we have the following problems:

- Transport (HTTP, SMTP, FTP)
- Core XML
- Partial implementation or confusing integration of SOAP specification

6.1.1.1 Transport problem

The transport used for web services is very important. HTTP represents the most popular transport for RPC calls via SOAP. Additionally, the embedded server that we will use for the prototype is HTTP based. This means that a reliable interoperability must exist between HTTP and SOAP.

A simple but widely popular example of HTTP interoperability problem relates to the usage of « SOAPAction » under SOAP 1.1. The SOAP 1.1 specification says this about the HTTP SOAPAction header:

> The SOAPAction HTTP request header field can be used to indicate the intent of the SOAP HTTP request. The value is a URI identifying the intent. SOAP places no restrictions on the format or specificity of the URI or that it is resolvable. An HTTP client MUST use this header field when issuing a SOAP HTTP Request.

The presence and content of the SOAPAction header field can be used by servers such as firewalls to appropriately filter SOAP request messages in HTTP. The header field value of empty string ("") means that the intent of the SOAP message is provided by the HTTP Request-URI. No value means that there is no indication of the intent of the message; ie the value of SOAPAction must be in quotes, unless it is a ‘‘null’’ value!

The most often encountered problem is the following: if a server requires that « SOAPAction » with ‘‘null’’ value, some clients will not be able to solve that case because all API client HTTP cannot define headers a ‘‘null’’ value! In this case, there are two possible solutions:

- **Re-implement API clients (an actually difficult task)**
- **And/or make sure that none of the servers requires a « SOAPAction » with a ‘‘null’’ value.**

This problem is nonexistent under SOAP 1.2 because a new code HTTP (427) was introduced and submitted to l’IANA to show the client that the server application requires a « SOAPAction » replaced with an optional parameter « action ».
See W3C/SOAP 1.2 specification for more details.

6.1.1.2 XML Problem

The second type of interoperability problems relates to XML analysis and the management of XSD schemas. Basically, SOAP uses XML and XML schemas; so its interoperability depends on both interoperabilities.

An interesting example of interoperability problem involving both XML analysis and HTTP transports is the Byte Order Mark (BOM). When data is sent

through HTTP, it is possible to indicate the data encoding – such as UTF-16 or UTF-8 – in the header Content-Type. It is also possible to state the coding of a XML fragment by inserting a set of bytes. When UTF-16 is sent, BOM is necessary regardless of the presence of the encoding field " Content-Type " in the header; but it is not the case with UTF-8.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

n++<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="http://soapinterop.org/"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/en
       coding/">
      <tns:echoStringResponse>
         <Return>string</Return>
      </tns:echoStringResponse>
   </soap:Body>
</soap:Envelope>
```

Code 36: XML problem

The first three characters are hexadecimals for "BOM UTF-8" , but as you can see, Content-Type is part of. Some implementations send the BOM for UTF-8 even if they are not needed. Others cannot process XML with BOM. The solution is to send it only when necessary and to manage it correctly. In fact, it is essential to well manage BOM for the processing of UTF-16 messages because it is a requirement in this situation. There is no magic solution to solve that problem. But once they are identified, it is better to use the exact specifications (generally available with W3C) that describe all standards and apply them scrupulously.

6.1.1.3 SOAP protocol problem

It is easy to work with SOAP. It required messages to be placed in an envelope with the content of the message included in an element of the body. Unfortunately, the inaccuracy of some specifications often represents the origin of the problems. Thus, it is sometimes difficult to figure out the real outcome. In other words, the diverse interpretation of the specifications yields to various implementations.

SOAP actually suggests optional elements like headers and gives way to a vast array of opportunities for body element constituents.

A usual problem linked to specification is the usage and interpretation of the optional SOAP header. The attribute "mustUnderstand" with its value specifies if the SOAP header is optional or compulsory. In the instance that the attribute

"mustUnderstand" defined as "1" or "True" depending on the SOAP version, the receiver must recognize the information displayed in the header. A problem already exists in the recognition of the header and more importantly in its processing to extract valued information. Another problem surfaces when this header must go through several intermediary servers. In this instance, each intermediary server removes what it needs and if necessary, adds more data for the next server to use. This logical inference still not clarified by specification is not yet implemented in API clients. It is now obvious the consequences of such inaccuracy should a very important task need to be executed by the last server

The list below represents a series of different interoperability problems. They are research archives.

http://groups.yahoo.com/group/soapbuilders
http://www.mssoapinterop.org/
http://www.xmethods.net/ilab/

As a general rule, the only way to avoid these kind of problems among a numerous cases if to make sure that the used API is big enough and that it has been proven effective on the Web. However, such problems will remain regardless of all efforts. The only known remedy is to conduct practical tests once the application is written and the service is well defined.

For our project, we are susceptible to these major problems. Therefore, we will follow these guidelines:

- Force to non null all optional attributes of SOAP header when necessary
- The encoding type will always be UTF-8 whether it is the declaration of XML prologue or in the http header. There is no reason to refer to UTF-16 since we are not dealing directly with objects.
- For this project, there is no processing between two Milegate communication. So, there is no need to worry about attribute "mustUnderstand" for SOAP 1.1.

6.1.2 Web Services Interoperability Organization (WS►I)

WS-I is an open industry organization chartered to establish and to document Best Practices for Web services interoperability.
It provides a Profile and Testing tools that can be used by web service community to aid in developing and deploying interoperable Web services.

Web Services Interoperability Organization is actually an organism created in 2002 by Microsoft, IBM, BEA and Intel with the purpose of reaching common specifications to all implementations of Web services. Today, there are more

than 130 members. It is recognized as the organization that governs and makes decisions about Web services standard.

Its first purpose is to insure implementations interoperability. To achieve that goal, it defined many profiles that each implementation must follow to be compatible with another implementation carrying the same profile. These profiles are sets of specifications to be used, and they indicate how to interpret these specifications.

6.1.2.1 WS-Profile description

Our research is based on the four most commonly used profiles

6.1.2.1.1 Basic Profile V1.1 (10 April 2006)

It is the base profile that every implementation must follow to adhere to the rules of compatibility with other implementations. The other three profiles depend on it.
The WS-I Basic Profile 1.1 provides interoperability guidance for basic SOAP messaging for Web services using SOAP, WSDL, etc.....

ISO/IEC 29361:2008 defines the WS-I Basic Profile 1.1, consisting of a set of non-proprietary Web services specifications, along with clarifications, refinements, interpretations and amplifications of those specifications that promote interoperability.
Due to its popularity, we will focus on the different constituents of that profile. Four compulsory modules and one optional module made up basic profile.

Transport : this is a transport protocol HTTP used for communication.

Reprssentation : this is a SOAP 1.1 protocol that allows to describe transmitted data.

Description : this is a descriptive language WSDL v1.0 that allows a (provider/supplier) to describe its exposed operations.

Discovery: this is a UDDI v2 protocol that allows to share services definitions.

Security (optional) : this is a HTTPS protocol that insure security during access and communications. It is optional because it is possible to encrypt the XML feed before it is included in HTTP.

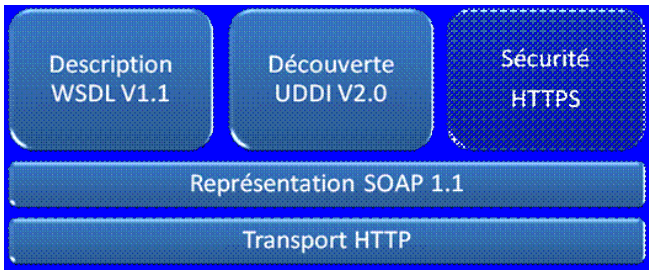


Illustration 37: modules of Basic Profile 1.1

6.1.2.1.2 Attachment Profile V1.0 (20 April 2006)

It is a profile that indicates the needed specifications to transmit large or binary data between components.
The Attachment Profile 1.0 complements the Basic Profile 1.1 to add support for interoperable SOAP Messages with attachments-based Web services.
ISO/IEC 29362:2008 defines the WS-I Attachments Profile 1.0, consisting of a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications that are intended to promote interoperability. It complements the WS-I Basic Profile 1.1 (ISO/IEC 29361:2008) to add support for interoperable SOAP Messages with Attachments-based Web services.

6.1.2.1.3 Simple SOAP Binding Profile V1.0 (24 August 2004)

This profile describes how to transport messages with HTTP protocole. It is very often used.
The Simple SOAP Binding Profile consists of those Basic Profile 1.0 requirements related to the serialization of the envelope and its representation in the message.
ISO/IEC 29363:2008 defines the WS-I Simple SOAP Binding Profile 1.0, consisting of a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications which promote interoperability.

6.1.2.1.4 Basic Security Profile V1.0 (30 March 2007)

It is a profile that defines security mechanisms required during Web services communications. There was no ISO specification found on this profile.

6.1.3 Presentation of a few frameworks

Before evaluating different framework, A brief explanation of SOAP is necessary.

6.1.3.1 What is SOAP?

SOAP (Simple Object Access Protocol) is a communication protocole between web services based upon the exchange of XML messages. This light data exchange protocole does not require any programming model and can be used in every communication style : synchronous or asynchronous, point to point or point to multipoint.

The SOAP specification recommended by W3C is divided in four parts:

- The SOAP envelope that defines a message contexte, its destination, its content and different options.
- The encoding SOAP rules defining data representation of an application in the body of a SOAP message (in particular its structure).
- A protocol (RPC) defining a series of requests and responses.
- The definition of the transport layer (http, smtp...) to be used to transport SOAP messages. It is usually referred to as SOAP message « encapsulation ».

The most recent version is v1.2 from April 27, 2007. However, many applications still use v1.1 of May 08, 2000, which is not a W3C recommendation but a specification resulting from researches conducted by DevelopMentor, IBM, Microsoft and UserLand Software.

6.1.3.2 SOAP encoding and formatting rules

A SOAP message is nothing else than an XML document made of an envelope that contains an optional header and a message body.

6.1.3.2.1 Encoding styles of SOAP messages

The specification defines two ways to format the XML messages within the body of the SOAP envelop:

- **Encoded** to mean that SOAP encoding uses a set of rules based on the XML Schema datatypes to encode the data, but the message doesn't conform to a particular schema. The set of rules is defined by the attribute « encodingStyle »

- **Literal** means data must be conformed to an XML schema or XSD (which is specified in the WSDL file) on the data attribute « type » or « element ».

  Type des messages SOAP
  There are two ways to structure a SOAP message. In the early versions of SOAP (before it was publicly published), SOAP was designed to support only RPC style. By the time the SOAP 1.0 spec was published, it was expanded to support both RPCs and unstructured messages (document).

- **RPC style** : All parameters are enveloped inside one named XML element. It means that encoded data in XML format are stocked in the body of the SOAP message before it is sent to the destination device. When using RPC style, the contents of the SOAP Body must conform to a structure that indicates the method name and contains a set of parameters. It looks like this:

```
<env:Body>
    <m:&methodName xmlns:m="someURI">
      <m:&param1>...</m:&param1>
      <m:&param2>...</m:&param2>
      ...
    </m:&methodName>
  </env:Body>
```

Code 38: SOAP RPC style

The response message has a similar structure containing the return value and any output parameters

- **Document Style**: sometimes the data to be send are already in XML. The document style is use in that case. Only the XSD rules can be used to encode or format the data.
  So when using Document style, you can structure the contents of the SOAP Body any way you like.

  Note: It is possible to do an "ACTION" as a document or as a parameter in a method called "ACTION".

DOCUMENT STYLE	RPC STYLE
<env:Body> <m:actionToDo xmlns:m="someURI"> xmlns:m="someURI"> ... </m: actionToDo> </env:Body>	<env:Body> <m:action <m: actionToDo> </m: actionToDo> </m: action> </env:Body>

Code 39: SOAP style comparison

The bigger difference is how to encode the message. In most circumstances, literal encoding is used with Document style and SOAP encoding with RPC style.
As it has been said above, literal encoding means that the Body contents conform to a specific XML Schema and SOAP encoding uses a set of rules based on the XML Schema datatypes to encode the data, but the message doesn't conform to a particular schema.
When using SOAP encoding, we would specify the element once and then reference the element as needed.
When using literal encoding, you would have to repeat this element each time it's referenced. So obviously it sounds like a good idea to use SOAP encoding. But, if we do, then we can't validate the message with an XML Schema, and we can't transform the message using XSLT.

Furthermore, RPC is the easiest approach for developers because it just needs the method name and the parameters of the remote object. Then SOAP RPC calls the method of a remote object and gives all needed parameters. SOAP layer serializes them within a XML stream, wrapped each stream in http or SMTP and finally sends the obtained packages to the server.
So SOAP RPC takes care of encoding and decoding messages, even for complex types and creates the SOAP proxy automatically.

In SOAP document type, SOAP layer sends a complete XML document to the server and does not wait for a feedback. The message might contain any type of XML data that can be slightly adapted to distant services. In document mode (document style encoding), the developer controls everything, from the choice of transport (HTTP, SMTP, etc....) to the serialization, including SOAP envelopes format.
Let us assume for a second that you are a developer and that you already have data in XML format. SOAP RPC also authorizes a literal encoding of XML data as a unique field. Since there is only one parameter – the tree XML that represents data – the SOAP layer has only one value to serialize. The SOAP layer also manages the transport, the "serialization/de-serialization" and the generation of proxys.

For "Document" model, it is a message in XML format. For"RPC" model, input-output arguments are encoded funtions under XML format.

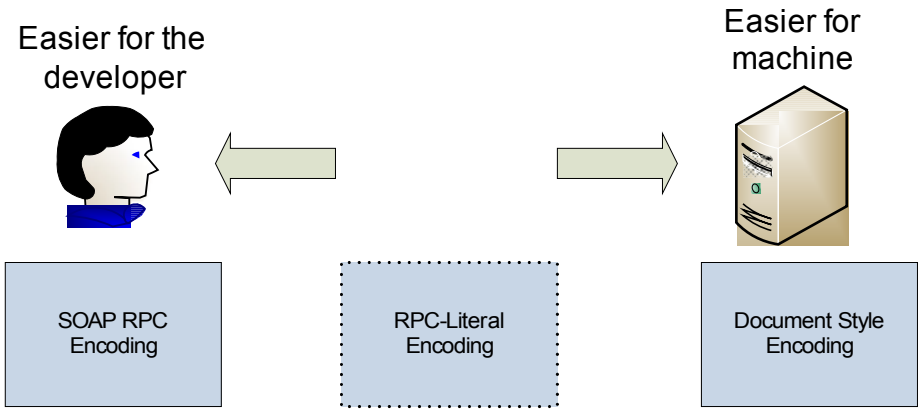


Illustration 40: suitable encoding type

Although SOAP RPC is simple to implement, this approach hardly rises up and does not perform well. Encoding SOAP RPC-literal requires more effort to the developer to handle XML parsing, but seek the least SOAP layer. It is therefore more difficult to implement, although it is more efficient and better care rises over SOAP RPC encoding.

Why SOAP RPC is easier to implement for the developer?:
Because with this type of encryption there is just to define the methods that will be part of the service. SOAP layer supports all the rest. In other words, you do not have to manually parse the XML tree to find necessary items.

Why document style is easier to implement for the system?:
The great advantage is that we know not only the XML tree (structure) to parse but we also have the opportunity to directly reach the data that interest us, while a general layer SOAP should be parsed the entire tree.
That reduces the number of processing system.
This type of message is very important for embedded systems.

6.1.3.2.2 KEYMILE: Constraints and Needs

KEYMILE does not allow queries directly the MileGate internal system; that implies we must necessarily go through the proprietary-XML provided. In addition the use of XML Schemas (present in the WSDL file) is essential here to validate the messages.
The whole system is composed of embedded components, it is necessary to consider the use of CPU resources.

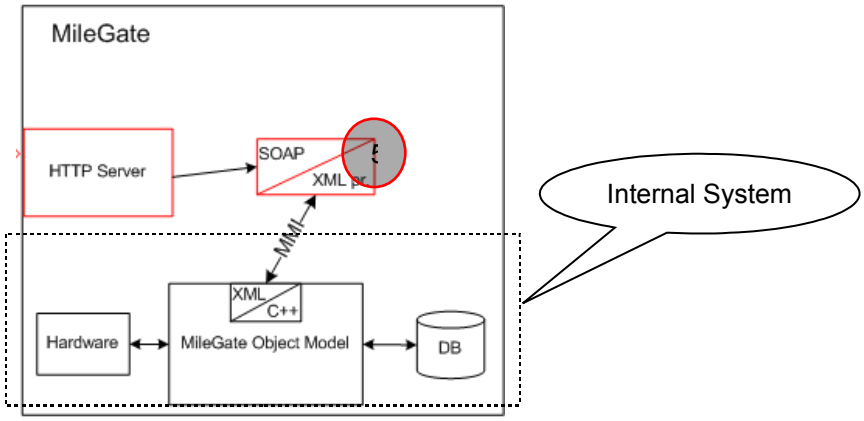


Illustration 41: KEYMILE Internal System

6.1.3.2.3 Combination and Interoperability list

The different different combinations are :

- RPC / Encoded
- RPC / Literal
- Document / Encoded
- Document / Literal

Knowing that the association between **Encoded** style and **Document** model is not implemented, and the norm **WS-I Basic Profile 1.0** chose **Literal** as standard instead of **Encoded** style, there are strong chances to recognize the following SOAP formats :

- RPC / Literal
- Document / Literal

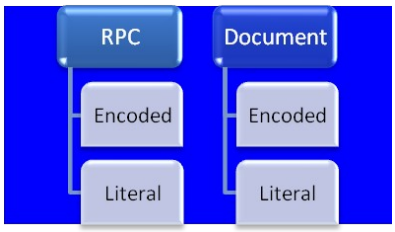


Illustration 42: Combination specification

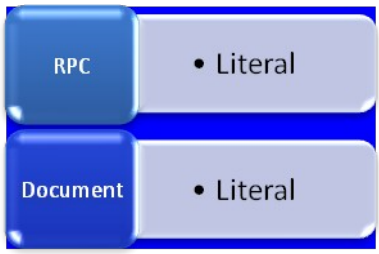


Illustration 43: Reality combination

6.1.3.3 Framework fonctionnalities

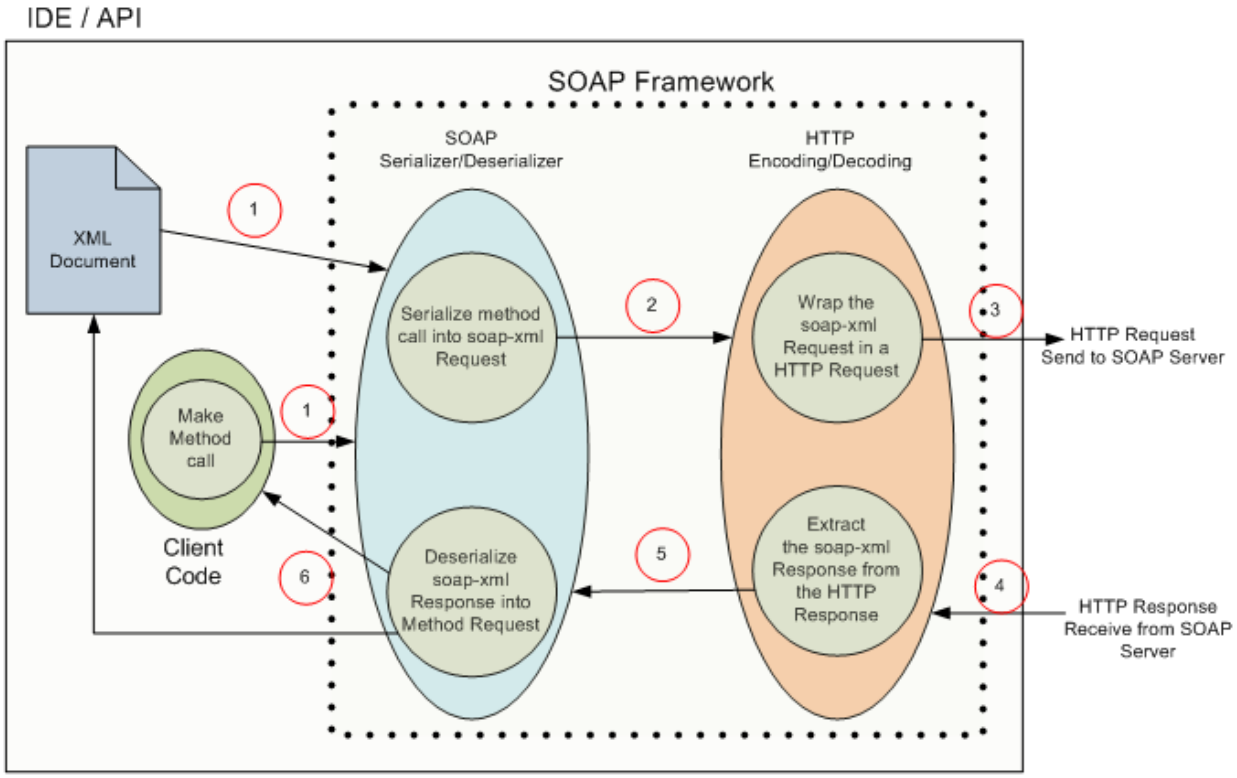


Illustration 44: What client Side framework should be allowed

Summary explanation of the diagram above:

The client code (written in various languages available like Java, C / C + +, Perl, Python, PHP ...) makes a service call by invoking the appropriate method in the SOAP package. We can already have data (RPC call, action to do) in XML format (1).

The SOAP package via SOAP serializer converts this invocation into a SOAP request and sends that to the HTTP encoder (2).
SOAP serialization converts (serializes) the public fields and properties of an object, or the parameters and return values of methods, into an XML stream that conforms to a specific XML Schema definition language (XSD) document.
SOAP serialization results in strongly typed classes with public properties and fields that are converted to a serial format (in this case, XML) for storage or transport.
As XML is an open standard, the XML stream can be processed by any application, as needed, regardless of platform.

It is a simple powerful concept (Serialization & De-serialization), which allows an object to retain its form even across a network.

The HTTP encoder wraps the SOAP message in a HTTP request and sends it to the SOAP server (3).
the SOAP server is simply an application server (appserver) running on the MileGate.

The response is received from the SOAP server by the HTTP encoder/decoder module(4) which decodes it and extracts the SOAP response which it hands to the SOAP deserializer (5).

The SOAP deserializer deserializes the message and gives the result to the client code (6) as the return value of the original invocation (1) or puts the response in XML Document format.

6.1.4 Framework evaluation

During the frameworks survey, we tried to find per programming language at least one framework that allows the use of web services. We chose these frameworks according to their popularity and/or according to the different features they offer.
The list below is not exhaustive and can be improved.

The following tools will be evaluated:

1. **AXIS 2**: It is a core engine for web services. It is one of the most popular frameworks of APACHE GROUP FOUNDATION. It allows to create (server) and to consume web services. One of its advantages is the availability of services (implementation) in C/C++ or in Java. So most features evaluated are available for AXIS2/JAVA and AXI2 / C.

2. **CXF**: is another APACHE GROUP FOUNDATION open source services framework that builds and develops web services using front-end programming APIs, like JAX-WS. It can only be used with java.

3. **WSO2** Frameworks: WSO2 is an innovative Open Source technology company devoted to in building Web services middleware. It focuses on providing modular open middleware for SOA powered by APACHE. WSO2 contains components that include utilities, modules (that can be dropped in to Apache Axis2/Java) and add-ons to other Apache Web Services projects.
WSO2 WSF (Web Service Framework) is the fully open source base framework. All WSO2 products are built on it. WSF implementation is available in C, C++, PHP, PERL, RUBY, PYTHON, JAVA, etc…

4. **METRO**: is an integrated stack that offers Web Services development by using Java Technology APIs and tools powered by SUN JAVA. Metro

stack is part of Project Metro and as part of GlassFish project. The Metro stack consists of JAX-WS, JAXB, and WSIT.

5. **WCF**: This is a unified programming model for Microsoft to generate service-oriented applications. It's based on .NET framework.
WCF applications can be developed using the different languages of Microsoft. NET applications such as Visual basic, C/C++, C#, java, etc…
While Microsoft is a board member of WS-I, it is not clear how many WS-I profiles they are committing to completely support.

We briefly considered other frameworks such as:

- gSOAP(C++)
- NuSOAP,Pear SOAP (PHP)
- SOAP ::Lite (Perl)
- PySAOP (python)

They allow to generate web services using SOAP and WSDL specification according to W3C. But most of them do not offer functionality as broad as WSO2. It is for that reason that we did not report them.

6.1.4.1 What to evaluate?

We have defined here a list of features that will be evaluated in the different frameworks. The features are:

- WS-Concepts
- Transport
- Encoding table
- Data binding
- General features

Important note: When a field is "NO" that means the evaluated field is not supported. And when the field is empty it means we have not found on the official website or projects associated any information about the evaluated field.

WS-Concepts

There are several specifications linked/associated to Web Services WS-*. Those specifications are in different levels of maturity, and are maintained by different standards organizations (OASIS, W3C, DMTF…). Each specification can be complemented or overlapped by another one or can even be in competition with another specification.

The purpose of this part is not to study each concept, but to provide the different concepts that can be supported by the various frameworks evaluated, and to identify the important concepts to our project according to the shortlist found.

WS-Concepts	AXIS2	CXF	WSO2 Frame-works	METRO	WCF
Addressing	✓	✓	✓	✓	✓
Transfer	✓ [A1]	NO	✓	NO±	✓
Management	NO *	NO	✓	NO	✓
Distribute Management	✓ [A2]	NO	✓	NO	✓
Notification	✓ [A2]	NO	✓	NO	✓
Atomic Transaction	✓ [A3]	NO	✓	✓	✓
Business Activity	✓ [A3]	NO	✓	NO	✓
Coordination	✓ [A3]	NO	✓	✓	✓
Eventing	✓ [A4]	NO	✓	NO	✓
Metadata Exchange	✓ [A5]	NO	✓	✓	✓
Reliable Messaging	✓ [A6]	✓	✓	✓	✓
Security	✓ [A7]	✓	✓	✓	✓

Table 45: WS-Concepts

[A1] : It is possible to use it via **xFer** (as an add-on module for Apache Axis2) which is a component of **WSO2 Commons**

[A2] : **MUSE** is an Apache project which is an implementation of the WS-ResourceFramework (WSRF), WS-BaseNotification (WSN), and WS-DistributedManagement (WSDM) specifications. It will be used as module.

[A3] : **Kandula** module implements WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity protocols based on Apache Axis and Axis2.

[A4] : **Savan/C** is a C implementation of WS-Eventing specification. We did not find a module for java…

Once the metadataExchange is engaged in Axis2, client can have the option to discover WSDL, XML Schema, and Policy in case a specific reference is passed to a client.

[A6]: WS-RM is given by the module **Sandesha** (implementations for the Apache Web Services project).

[A7]: **Rampart** is a module based on Apache WSS4J to provide WS-Security features. Apache WSS4J is an implementation of the OASIS Web Services Security from OASIS Web Services Security TC (SecureConversation; SecurePolicy, Trust…).

* :We did not find a module that is directly supported by axis2 but it is possible to bypass this problem by declaring the parameters needed directly in the WS-addressing header. Unfortunately the client must be known it to fix the problem. It will simply come down to pure guess.

± :It is not included in the available list presented on the official website. Nevertheless, the site displays and conducts tests that emphasize that parameter with other frameworks. Further research is needed.

Transport

We evaluate here the different types of data transfer protocols or communication protocols supported by the frameworks in a client – server communication. There will not be any details regarding the evaluated protocols. A simple and brief definition is available in the glossary and annex sections.

Transport	AXIS2	CXF	WSO2 Frame-works	METRO	WCF
HTTP/HTTPS	✓	✓	✓	✓	✓
SMTP	✓		✓	✓	✓
POP3	✓		✓	✓	
FTP	✓		✓		
TCP	✓		✓	✓	✓
JABBER	Exp.			NO	
JMS	✓	✓	✓	✓	
In-VM	✓	✓	✓	✓	
Proxy Support*	✓		✓	✓	✓

Table 46: Transport

* : Proxy is not a communication protocol. We only need to know if the requested SOAP included in any of those protocols can go through a server proxy..
Exp = Experimental.

Encoding table

It proposes a short list of different encoding mechanism associated with XML engine that can be supported by the evaluated framework.

Encoding Table	AXIS2	CXF	WSO2 Frame-works	METRO	WCF
XML Textual	✓	✓	✓	✓	✓
MTOM	✓	✓	✓	✓	✓
FastInfoset	✓	✓	✓	✓	✓
JSON	✓	✓	✓	✓	✓

Table 47: Encoding Table

Data binding

A list of few technologies for accessing XML by binding it to programming language types.

The proposed list provides a convenient way to process XML content using Java objects by binding its XML schema to Java representation.

Data Binding	AXIS2	CXF	WSO2	METRO	WCF
XMLBeans	✓	✓	✓		
Castor	✓		✓		
JiBX	✓		✓		
JAXB	✓	✓	✓	✓	

Table 48: Data Binding

General Features

This table contains a set of information with an overview of a framework.

General Features	AXIS2	CXF	WSO2	METRO	WCF
BP 1.1 [1]	✓		✓	✓	✓
AP 1.0 [2]	✓			✓	
SSBP 1.0 [3]	✓			✓	✓
BSP 1.0 [4]					✓
Open Source	✓	✓	✓	✓	NO
Soap 1.1	✓		✓	✓	✓
Soap 1.2	✓		✓	✓	✓
WSDL 1.1	✓		✓	✓	✓
WSDL 2.0	✓		✓	✓	✓
WSDL -> code server	✓		✓	✓	✓
WSDL -> code Client	✓		✓	✓	✓
Eclipse Plugins	✓		✓	✓	NO
NetBeans Plugins	NO		NO	✓	NO
IDEA Plugins	✓		✓	✓	NO
Hot Deployment	Axis2		Apache build tools	glassFish	Visual Studio

Table 49: General features

[1] : Basic profile Compliant
[2] : Attachment Profile Compliant
[3] : Simple SOAP Binding Profile Compliant
[4] : Basic Security Profile Compliant

6.1.4.2 Definitions

ADB:
short for Axis2 Data Binding Framework is probably the simplest method of generating an Axis2 client. In most cases, all pertinent classes are created as inner classes of a main stub class. ADB is very easy to use, but it does have limitations. It is not meant to be a full schema binding application, and has difficulty with structures such as XML Schema element extensions and restrictions.

Castor:
It is an open source-binding framework for moving data from XML to java programming language objects and from java to databases.

CIM:
The Common Information Model is an open standard that defines how managed elements in an IT environment are represented as a common set of objects and the relationships between them. This is intended to allow consistent management of these managed elements, independent of their manufacturer or provider.
The CIM standard is defined and published by the Distributed Management Task Force (DMTF).
For more information see DMTF specifications.

FAST INFOSET:
It is a coding method of XML document into a binary data format developed by SUN and adopted today as ISO standard. Its purpose is to decrease not only the size of XML documents, but also the necessary resources for parsing document analysis.

JABBER:
short for JavaScript Object Notation, It is a lightweight data-interchange format very easy for reading and writing by humans, but also easy for machines to parse and generate.
The JSON format is often used for serialization, transmitting structured data over a network connection. Its main application is in Ajax web application programming, where it serves as an alternative to the use of the XML format.

JAX-WS:
The Java API for XML Web Services is a Java programming language API for creating web services. It is part of the Java EE platform from Sun Microsystems. Like the other Java EE APIs, JAX-WS uses annotations introduced in Java SE 5 to simplify the development and deployment of web service clients and endpoints.

JAXB:
Java Architecture for XML Binding (JAXB) allows Java developers to map Java classes to XML representations. JAXB provides two main features: the ability to serialize Java objects into XML and the inverse, i.e. to de-serialize XML back into Java objects. In other words, JAXB allows to store and to retrieve data from memory in any XML format without the need of implementing a specific set of XML loading and saving routines for the program's class structure.

JiBX:
It is a full data-binding framework that actually provides not only WSDL-to-Java conversion, as covered in this document, but also Java-to-XML conversion. In some ways, JiBX provides the best of both worlds. JiBX is extremely flexible, enabling you to choose the classes that represent your entities, but it can be complicated to set up.

JSON:
short for JavaScript Object Notation, It is a lightweight data-interchange format very

easy for reading and writing by humans, but also easy for machines to parse and generate.
The JSON format is often used for serialization, transmitting structured data over a network connection. Its main application is in Ajax web application programming, where it serves as an alternative to the use of the XML format.

MTOM:

It is the W3C Message Transmission Optimization Mechanism, a method of efficiently sending binary data to and from web services. For more details see the W3C specification.

OASIS:

(Organization for the Advancement of Structured Information Standards) is a not-for-profit consortium that drives the development, convergence and adoption of open standards for the global information society. The consortium produces more Web services standards than any other organization along with standards for security, e-business, and standardization efforts in the public sector and for application-specific markets.

XML Textual:

In XML textual mode, it is possible to see and to modify XML document. A direct access exists, and it is easy to index the document in comparison to XML binary mode.

XMLBeans:

Unlike ADB, XMLBeans is a fully functional schema compiler; so it doesn't carry the same limitations as ADB. However, it is a little more complicated to use than ADB. It generates a huge number of files; and the programming model, while certainly usable, is not as straightforward as ADB.

WSIT:

Web Services Interoperability Technology is an open-source project started by Sun Microsystems to develop the next-generation of web service technologies. It consists of Java programming language APIs that allow developers to create web service clients and services that interoperate between the Java platform and Microsoft's Windows Communication Foundation (WCF) and .NET.

WSIT implements the following WS-Concepts:

Messaging	Metadata	Security	Transaction
• WS-ReliableMessaging • WS-RMPolicy	• WS-MetadataExchange • WS-Transfer • WS-Policy	• WS-Security • WS-SecureConversation • WS-Trust • WS-SecurityPolicy	• WS-Coordination • WS-AtomicTransaction

WCF: Short for **W**indows **C**ommunication **F**oundation.

6.1.5 Tests tools

The WS-I offers two tools for interoperability which are real references. These include: Monitor & Validator (Analyzer). These two tools are combined in the Interoperability Testing Tools 1.1.
"Interoperability Testing Tools 1.1 is designed to help developers determine whether their Web services are conformant with WS-I profile guideline."[29]

6.1.5.1 Monitor

Monitor allows to capture the SOAP messages exchanged between a consumer and producer services.
It behaves like a proxy server with in / out ports, and as the messages go through it, the monitor completes a trace file.

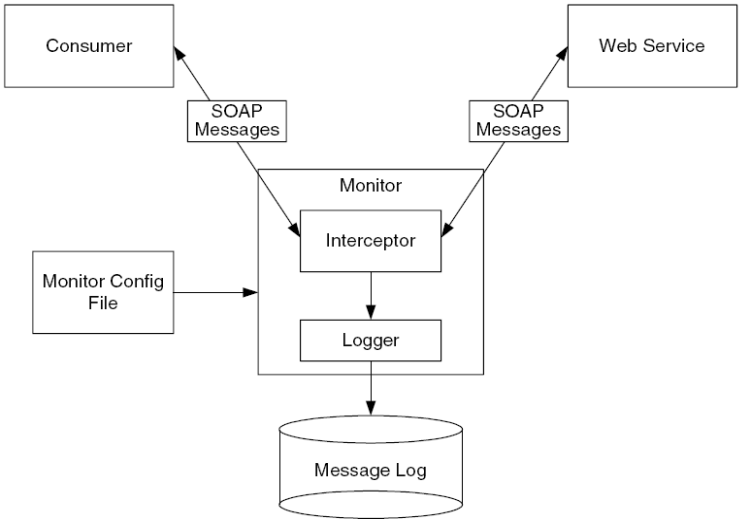


Illustration 50: WS-Monitor

6.1.5.2 Validator

Validator generates a report of compliance of the WSDL contract and SOAP exchange message which are transmitted in regard of WS-I specifications Profiles. The compliance report is based to Basic Profile 1.1 and SSBP 1.0 and the message log coming from Monitor sniffing.

[29] http://www.ws-i.org/

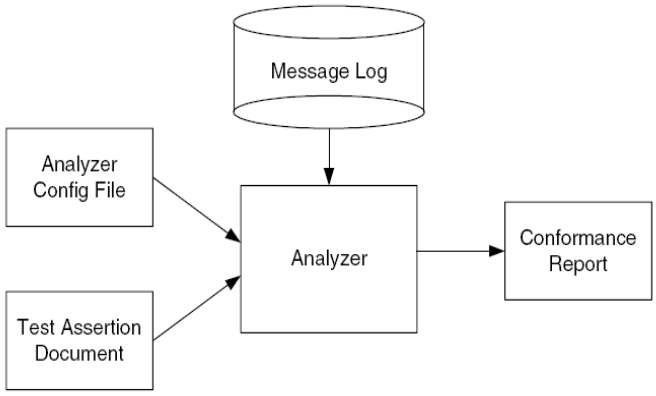


Illustration 51: WS-Validator

The following diagram shows a much more complete picture of interoperability test tool 1.1

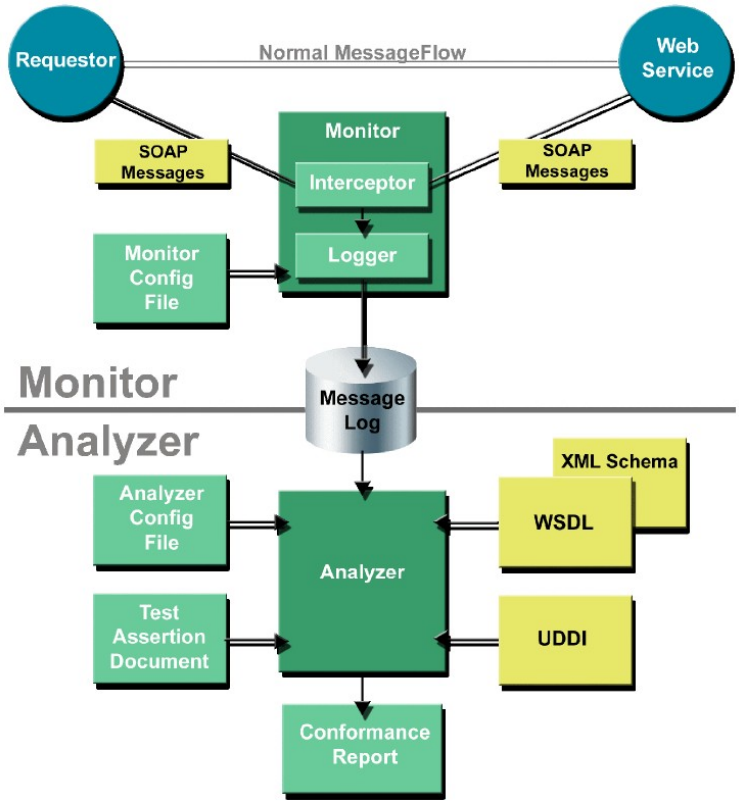


Illustration 52: Interoperability Testing Tools 1.1

While WS-BP will ensure that we use the same levels of specifications, it must still ensure that the types exchanged are well represented in each environment, in our case with the client. This involves a further problem of interoperability which is the spread of data from one end to another.

Indeed there are some types that are specific to a given framework such as the DataSet of .Net, which has no equivalent in Java.

.Net ⇔ W3C ⇔ Java

.Net	WSDL	Java
string	xs:string	java.lang.String
int	xs:int	java.lang.Integer

Illustration 53: Exemple of exchange Scenario of simple type

Unfortunately, the time available will not allow us to study it.

7 Realization of the Prototype

Abstract

This chapter describes the functionality of our prototype and describes the different stages the information runs trough. One this stages, the generation of the WSDL using an stylesheet transformation described more in detail and shows the added value to the Web Service description.

7.1 Flow of information

For the complete flow of information we have to remember the the Web Service Illustration which shows a global view of the system. The different steps are here described briefly.
Additional information can be found under the mentioned references.

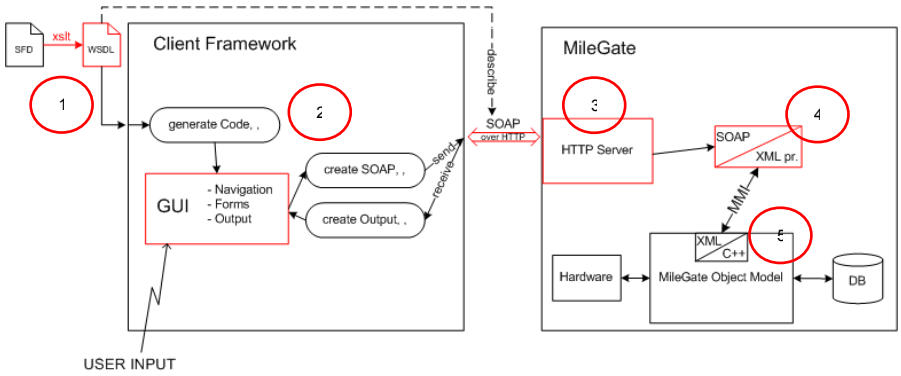


Illustration 4 ”Approach with web service” from chapter 1 paragraph “Work to perform”

7.1.1 SFD to WSDL

1

The description of the Web Service needs finally be generated automatically from files called “SFD”. Those files are provided by the system and contain almost all the necessary information for our description.

The description (WSDL file) we provided, is written manually and considers just one single interface. The complete system has more than hundred interfaces where the operations need to be accessible.

As the SFD file is written in XML, a transformation stylesheet (XSLT) will be used. The transformation from SFD to the Accesspoint Definition File ADF performs such a transformation and builds the basis for the transformation to WSDL.

See section 7.2 for more information about the transformation.

7.1.2 WSDL to Code & SOAP

2

The generation of the code is the job of the framework. Almost every conventional programming language provides at least one framework.
With the import of the WSDL-file, the operations defined in the description were made accessible for the programmer.

Illustration 54: SOAPui stubs

Generation of the SOAP messages is depending on the programming language but the skeleton is provided in the description.

On the right hand side we can see the automatically generated tree of the program SOAPui. The interface (wsdl:portType) containing all the operations (wsdl:operation) defined in the WSDL file.
The SOAP request contains the defined message elements of the selected operation as header or body content.

7.1.3 HTTP Server

3

The task of the HTTP Server is to extract the SOAP message from the HTTP request and deliver it to the system. We see in the following code section the SOAP message transported as a HTTP POST method. The SOAP message is identical and just added as payload.

```
POST /wsman/ HTTP/1.1
Content-Type: application/soap+xml;charset=UTF-8;action="http://schem-
as.xmlsoap.org/ws/2004/09/transfer/Get"
User-Agent: Jakarta Commons-HttpClient/3.1
Host: localhost:9357
Content-Length: 1924

<!--



    GET LABEL



-->
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
```

```
    xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml"
    xmlns:wsa  ="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
    <soapenv:Header>
        <wsa:To>http://192.168.32.171/wsman</wsa:To>
        <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
        <wsa:ReplyTo>
            <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/rol
              e/anonymous</wsa:Address>
        </wsa:ReplyTo>
        <wsman:SelectorSet>
            <wsman:Selector Name="mf">main</wsman:Selector>
            <wsman:Selector Name="proerty">www.keymile.com/mg/2008/06/MoInfo
              /Label</wsman:Selector>
        </wsman:SelectorSet>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
        </wsa:Action>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued
        </wsa:MessageID>
    </soapenv:Header>
    <soapenv:Body/>
</soapenv:Envelope>
```

Code 55: Wireshark HTTP capture

The HTTP server responses this request with an "HTTP/1.1 200 OK". The payload of this HTTP reply is the requested information as a SOAP message.
The source code of the HTTP server "Baracuda" had not been modified and will not be added as annexe (under NDA).

7.1.4 SOAP to KOAP

4

We have a SOAP request arriving at the MileGate which needs to be answered by the system. The only interface to the embedded system needs KOAP requests (see Code 12: KOAP request) which have exactly the same body as our SOAP requests.
A simple DOM parser identifies the addressed unit and function and changes the syntax of the message (SOAP to KOAP and vice versa).

The SOAP and KOAP encoder provided by KEYMILE has not been modified, the source code is under publishing restriction.

7.1.5 KOAP to C++

5

The invocation of the C++ routines is used for all different kinds of configuration interfaces. Web Service can use the same manner as we changed before from SOAP to KOAP messages.
The architecture of the MileGate is hidden behind the KOAP message.

Source code under publishing restriction.

7.2 WSDL File generation

For this project we had to modify the existing XSLT provided by the KEYMILE according to the new parts of the manual described the Web Service.

The changes of the XSLT result directly all the parts added to the WSDL description. Thus the modifications of the XSLT will not be commented, we have to mention at this point the added value for the WDSL file.

The new parts added to the WSDL file are:

- A new fault type has been declared
- The header & fault messages had been declared
- The fault message had been integrated into each operation
- Declaration of WS-Transfer "soapAction" for each operation
- The automatic binding of the headers into the SOAP messages had been added for each operation
- The wsdl:service element with the Endpoint References (addressing of the resource and management function) had been add

This list corresponds to the highlighted parts in the manually described Web Service in annexe:WSDL.

In the XSLT file provided as annexe: XSLT, all the newly added parts were highlighted.

8 Tests

Abstract

This chapter contains the definition of the tests and the validation of the functionality of the prototype.

8.1 Tests Definition

This section contains the definitions for the tests we want to perform on the prototype.

8.1.1 Verification of the Web Service

The verification task for the Web Service is very important but in this case also quite difficult because the reaction of the MileGate system is predefined. We grouped the verification into two major parties. The first part is the validation which checks if the descriptions follow the standards.
The second part contains some basic tests of the system. Here we have to be aware that for lot of tests the existing software is involved which will not be modified at the moment.

8.1.1.1 Validation

For the validation of the Web Service, the most important point is that the description follows the rules defined for WDSL. With the XSLT we generate at the moment just the description for the definition WSDL 1.1. We wont validate WSDL 2.0 because the SOAP to KOAP translation in the MileGate does not support WSDL 2.0.

Also the SOAP messages need to be in accordance with the standard. This is difficult to test at the moment because the final SOAP request is generated by the client framework.

The header fields are defined according to the used standards (WS-Management and WS-Addressing) and included automatically into the SOAP message skeleton. The used Namespaces of SOAP and for the two Web Service concepts are also written automatically into the message.
For the body part of the SOAP message, the elements defined in the WSDL are included.

The validation of the Web Service functions will be performed with SOAPui 2.5.1, a Web Service Testing tool developed by eviware.
The following example shows the automatic generated SOAP message of the program SOAPui[30] (Web Service Testing Tool) of the operation 'SetLabel__operation' defined in the WSDL file. The addressing parameters are missing because the endpoint reference need to be selected by programming.

[30]http://www.soapui.org

```
<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 xmlns:add="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:man="http://schemas.xmlsoap.org/ws/2005/06/management"
 xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml">
    <soap:Header>
        <add:MessageID/>
        <add:Action/>
        <add:ReplyTo/>
        <add:To/>
        <man:SelectorSet/>
        <man:ResourceURI/>
    </soap:Header>
    <soap:Body>
        <mob:Label>
            <mob:user>?</mob:user>
            <mob:service>?</mob:service>
            <mob:description>?</mob:description>
        </mob:Label>
    </soap:Body>
</soap:Envelope>
```

Code 56: SOAPui skeleton

Defined endpoint reference in the WSDL file:

```
<wsa:EndpointReference name="labelEPR"
 xmlns:wsaw="http://www.w3.org/2006/02/addressing/wsdl">
    <wsa:Address>http://localhost:9357/wsman/</wsa:Address>
    <wsa:ReferenceParameters>
        <wsman:SelectorSet>
            <wsman:Selector name="mf">main</wsman:Selector>
            <wsman:Selector name="property">/Label</wsman:Selector>
        </wsman:SelectorSet>
        <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
    </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

Code 57: WSDL endpoint reference

With the parameters from the endpoint reference the request looks as followed (the EPR has to be added in the framework):

```
<soap:Envelope
 xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 xmlns:add="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:man="http://schemas.xmlsoap.org/ws/2005/06/management"
 xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml">
    <soap:Header>
        <add:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued</add:Mes
              sageID>
        <add:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
        </add:Action>
        <add:ReplyTo>
           <add:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role
              /anonymous</add:Address>
```

```
        </add:ReplyTo>
        <add:To>http://localhost:9357/man</add:To>
        <man:SelectorSet>                                          } from EPR
           <man:Selector name="mf">main</man:Selector>
           <man:Selector name="property">/Label</man:Selector>
        </man:SelectorSet>
        <man:ResourceURI>/unit-12</man:ResourceURI>
    </soap:Header>
    <soap:Body>
        <mob:Label>
            <mob:user>?</mob:user>
            <mob:service>?</mob:service>
            <mob:description>?</mob:description>
        </mob:Label>
    </soap:Body>
</soap:Envelope>
```

Code 58: merged SOAP request

- SelectorSet and ResourceURI are specified in endpoint reference EPR
- Action is specified in the <wsdl:operation>
- MessageID and ReplyTo must be added with the framework

8.1.1.2 Testing

The testing does not completely verify if the Web Services is functioning perfectly. Testing of the function has to be verified with a framework. The aim of this part is to document the reactions on certain requests and to suggest some modifications for the actual implementation.

We want to check the reaction on malformed addressing (unit, mf, property), malformed format of the body and of course also the reaction on a well formed request. Additionally we want to check:

- Same MessageID
- No address
- Malformed ResourceURI (unit)
- Malformed Selector (mf and property)
- Malformed SOAP body

8.1.2 Framework verification

It is important to note that a basic prototype is already realized. We can generate and send SOAP requests manually. This proves the feasibility of the service described.

For more comfort we will extend the completion of the prototype by using a framework other than SoapUI. So we can test our service under a condition of actual use. To do this we need to install a framework.

Our choice is focused on AXIS2 since it is the only one framework we have studied in depth. For security we will also install WCF (Visual Studio).

The installation of an http server is no longer necessary because we already have one available (baraccuda).

8.1.2.1 Framework Installation

AXIS2 :

- Installation of fedora 10
- Installation of a suitable container such as Tomcat for linux.
- Installation of xFer via ant (xFer module is needed for WS-Transfer)

WCF : Installation of visual studio.

8.1.2.2 Test Definition

The following tests will be done at possible:

- Generate java classes from our wsdl file.
- Test the integration of xFer module (ws-transfer) with the sample code provided
- Test the reaction of the framework opposite to WS-Management may be it supported under WS-addressing.
- Send a SetLabel an receive a GetLabel.
- Interoperability testing using the WS-I tools (WS-Validator & WS-Monitor) and see the conformance report.
- Use WSO2 Carbon tools to convert WSDL 1.1 to 2.0 which is the real recommendation of W3C and check if the service can be consume too with the new wsdl file.

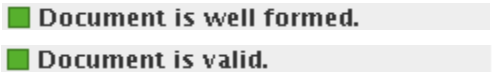
8.2 Validation of performed tests

This section contains the validation of the tests we performed on the prototype.

8.2.1 Validation of files / messages

The validation of the WSDL files was performed with the "<oXygen/> XML Editor 8.2".
The manual described WSDL file MILEGATE.wsdl and the generated (XSLT) file mob_mainbase_xml.wsdl and mob_mainequipment.wsdl had been validated for WSDL 1.1 and checked if the XML is wellformed with oXygen.

All this tree files are have the result:



WSDL 1.1 validation successful

The exchanged SOAP messages has been checked if they are wellforemed and correspond to the XML schema (http://schemas.xmlsoap.org/soap/envelope) of SOAP 1.1. SOAPui supports only SOAP 1.1. This verification has been made for all the SOAP messages (request and response) described in this chapter.

SOAP 1.1 check successful

8.2.2 Testing the response of the MileGate

The first test is the confirmation of the functionality based on a well-formed SOAP request. Later we observe the reaction on malformed requests and make some suggestions.
As described in the definition of the test, the SOAP request is just partially generated by SOAPui. The endpoint references were added manually.

8.2.2.1 Well-formed request

Request (GetLabel) sent with SOAPui:

```
<!--


    GET LABEL


-->
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml"
    xmlns:wsa  ="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
    <soapenv:Header>
        <wsa:To>http://192.168.32.171/wsman</wsa:To>
        <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
        <wsa:ReplyTo>
            <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/rol
              e/anonymous</wsa:Address>
        </wsa:ReplyTo>
        <wsman:SelectorSet>
            <wsman:Selector Name="mf">main</wsman:Selector>
            <wsman:Selector Name="proerty">www.keymile.com/mg/2008/06/MoInfo/
              Label</wsman:Selector>
        </wsman:SelectorSet>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Get
        </wsa:Action>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued
        </wsa:MessageID>
    </soapenv:Header>
    <soapenv:Body/>
</soapenv:Envelope>
```

Code 59: GetLabel request

Response of MileGate:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
    <env:Header>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued-
                    resp</wsa:MessageID>
        <wsa:RelatesTo>urn:uuid:d2345623-bc89-4323-9e83-ueldj
                    fued</wsa:RelatesTo>
        <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonym
                    ous</wsa:To>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/GetRe
                    sponse</wsa:Action>
    </env:Header>
    <env:Body>
```

```
        <Label>
            <user>?</user>
            <service>?</service>
            <description>?</description>
        </Label>
    </env:Body>
</env:Envelope>
```

Code 60: GetLabel response

Request (SetLabel) sent with SOAPui:

```
<!--


    SET LABEL


-->
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:mob="http://www.keymile.com/milegate/ws/mob_mainbase_xml"
    xmlns:wsa  ="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd">
    <soapenv:Header>
        <wsa:To>http://192.168.32.171/wsman</wsa:To>
        <wsman:ResourceURI>/unit-11</wsman:ResourceURI>
        <wsa:ReplyTo>
            <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/rol
              e/anonymous</wsa:Address>
        </wsa:ReplyTo>
        <wsman:SelectorSet>
            <wsman:Selector Name="mf">main</wsman:Selector>
            <wsman:Selector Name="property">
              www.keymile.com/mg/2008/06/MoInfo/Label</wsman:Selector>
        </wsman:SelectorSet>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Put
        </wsa:Action>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued
        </wsa:MessageID>
    </soapenv:Header>
    <soapenv:Body>
        <mob:Label>
            <mob:user>user</mob:user>
            <mob:service>service</mob:service>
            <mob:description>description</mob:description>
        </mob:Label>
    </soapenv:Body>
</soapenv:Envelope>
```

Code 61: SetLabel request

Response of MileGate:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
    <env:Header>
        <wsa:MessageID>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued-resp
        </wsa:MessageID>
        <wsa:RelatesTo>urn:uuid:d2345623-bc89-4323-9e83-ueldjfued
        </wsa:RelatesTo>
        <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonym
              ous</wsa:To>
        <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/PutRe
              sponse</wsa:Action>
    </env:Header>
    <env:Body></env:Body>
</env:Envelope>
```

Code 62: SetLabel response

If we send the first request another time, we get the new values from the "SetLabel" request:

```
    ...
    <env:Body>
        <Label>
            <user>user</user>
            <service>service</service>
            <description>description</description>
        </Label>
    </env:Body>
    …
```

Code 63: new GetLabel response

Comment:
The request has some problems if the comment at the beginning of the request is removed. This error is lied to problems of the HTTP server implementation which has difficulties with the handling of too small requests. Will be solved on the next version of the HTTP server on the MileGate.

All the requests were replied successful and in a response time of between 7ms and 24ms (10 tries).

Validation successful

8.2.2.2 Malformed request

Same MessageID

The system does not react on the message identities, it just adds the string "-resp" to the response. It is basically the job of the client programmer to ensure that the ID is unique.

A check mechanism with timer would be possible on server side which generates the predefined WS-Addressing fault wsa:DuplicateMessageID.

No address

The system does not react on a missing or mismatching address (wsa:To) in the SOAP message. This is field is actually not necessary because the address on the HTTP layer is defined with the service declaration of the interface. The principal aim of this field is to allow the forwarding of the message to another system endpoint.

If this will be implemented in the future, the match of the wsa:To and the local address should be verified. In case of success, the request should be treated, otherwise it should be forwarded or an wsa:MissingAddressInEPR returned.

Malformed ResourceURI (unit)

The system responds with the fault "Operation not found"

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
    <env:Body>
        <env:Fault>
            <env:Code>
                <env:Value>env:Sender</env:Value>
            </env:Code>
            <env:Reason>
                <env:Text xml:lang="en">
                    EXTERNAL.PLATFORM.MOB.OPERATION_NOT_FOUND</env:Text>
            </env:Reason>
        </env:Fault>
    </env:Body>
</env:Envelope>
```

Code 64: Response on malformed ResourceURI

This reaction provides the programmer the information that something with the addressing went wrong. The wanted function is not available for this resource. We suggest either to indicate the unavailability of the operation for this resource or to use the predefined wsa:InvalidEPR to keep it general.

Malformed Selector (mf and property)

The reaction of the system is for both selectors identically to the malformed ReourceURI (EXTERNAL.PLATFORM.MOB.OPERATION_NOT_FOUND).

Here it is also possible to return an wsa:InvalidEPR fault or to use something more specific as "unsupported/unknown operation".

Malformed SOAP body

With a malformed body, means that it does not correspond the WSDL description. Actually, the service does not know the WSDL description but it has all the information provided in the ADF (accesspoint definition file) and verifies the syntax there.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
    <env:Body>
        <env:Fault>
            <env:Code>
                <env:Value>env:Sender</env:Value>
            </env:Code>
            <env:Reason>
            <env:Text xml:lang="en">
                    EXTERNAL.PLATFORM.MOB.XML_DECODING_ERROR</env:Text>
            </env:Reason>
        </env:Fault>
    </env:Body>
</env:Envelope>
```

Code 65: Response on malformed SOAP body

The provided error description "xml decoding error" does not clearly indicate the reason and should be more precise.

We have the same reaction if we delete one element of a valid body.

Remark

Most of the suggestions will be difficult to implement because the system does not provide further information about the failure. The reaction on "same messageID" and "No address" can be responded by the service which transformates the SOAP to KOAP messages. The other reactions need provoke a error message in the base system which gives information about the internal structure away.

8.2.3 Validation of framework

8.2.3.1 Problems encountered

When we try to generate the client code from the WSDL file by using the program WSDL2JAVA of AXIS, we get lot errors. at first we thought that the error was internal to the AXIS2 framework as it has been said on the fedora forum[31]. But by trying another test tool[32] to validate the wsdl file, we realized it did not conform to the W3C specification.

Error 1

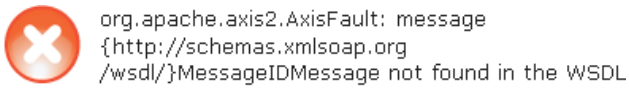


Illustration 66: Error 1

Error 2: after remove MessageIDMessage

Illustration 67: Error 2

We try to convert the WSDL file 1.1 to 2.0 too and get the same problem. That means the WSDL file contains statements that are not conforming to W3C specification symbolize by "http://schemas.xmlsoap.org " in the message error.

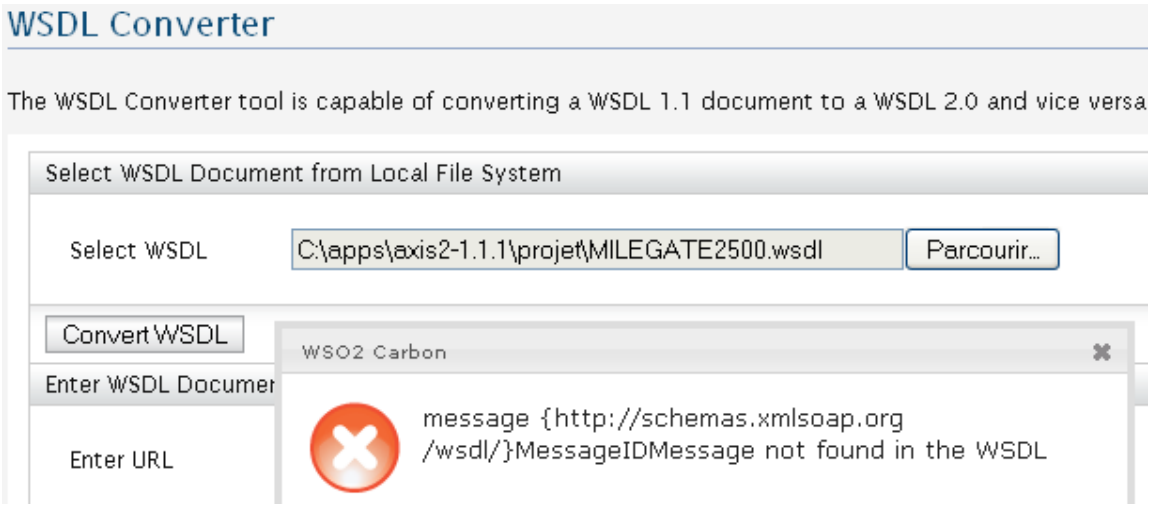


Illustration 68: WSDL Converter

[31] http://forums.fedora-fr.org/

[32] http://wso2.org/tools

To be sure of our test, we take the Axis2UserGuide.wsdl then we do the same test and the results below is obtained:

1. Conversion of WSDL 1.1 to WSDL 2.0 → Successful
2. Generation of java class with WSDL 1.1 and WSDL 2.0 →Successful (we don't waste time to try to use those classes because we don't need them)

Furthermore we load the xFer module according to WSO2 tutorial and test it with the sample client provided by WSO2.

We get a connectException as report below :

```
Exception in thread "main" org.apache.axis2.AxisFault: Connection timed out:
connect
        at org.apache.axis2.AxisFault.makeFault(AxisFault.java:431)
        at org.apache.axis2.transport.http.HTTPSender.sendViaPost(HTTPSender-
.java:195)
        at
org.apache.axis2.transport.http.HTTPSender.send(HTTPSender.java:77)
        at org.apache.axis2.transport.http.CommonsHTTPTransportSender-
.writeMessageWithCommons(Common
sHTTPTransportSender.java:328)
        at
org.apache.axis2.transport.http.CommonsHTTPTransportSender.invoke(CommonsHT-
TPTransportSen
der.java:206)
…
…
…
```

Code 69: connectException

We fix that error by setting properly the AXIS2_HOME variable to point to the location in which we've unpacked it…. This problem is simply caused by the environment variable…

According to the remaining time, we could not perform all tests necessary for a proper validation of the prototype.

It is important to note that while the java classes are not generated most of the tests are not possible.

9 Conclusion

The study of the SOA was very interesting and helped to understand the advancement of the Internet in direction of Web Services. Not all the ideas are implementable for the MileGate because we have existing constraints and what is even more important, an existing and functional system. As all the transformations towards an service oriented architecture, the process will be very time-consuming.

Web Services have some exceptional concepts which offers a mass of new possibilities. Here a careful study of the requirements and on the functionalities wanted to offer had to be performed.
Attention have to be paid on the level of complexity of the system to not set limits for the implementation on the client side but also for not defining it vague or ambiguous.

"Things should be made as simple as possible, but no simpler."
Quote Albert Einstein

The endpoints of the Web Service and its management pose some problems which could be solved with the different techniques described. The easiest way to manage them is to use the endpoint references EPR by programming on the client side. A adapted version of the 'discover' (MileGate operation) which furnishes just the required information for the Web Service would be more efficient and would additionally allow to hide the infrastructure from the client.

We have a wide range of embedded HTTP server which offers C/C++ using as KEYMILE hopes. However, a deeper study of the use of resources and memory is still necessary

The actual MileGate notifications, which follows the same principles as WS-Notification, should be translated into Web Service notifications and described with meta description to make it machine-readable.

For purposes of flexibility, the direct access of the management functions (not over two parameters, e.g. main/label). It will be easier to define constraints for the invocation of operations which are related to the access address (EPR). The further idea is that we need to ensure that just possible functions can be invoked. Possibilities therefore are the simple response with an error, the 'ValidWhile' provided by WS-A Metadata or the use of relation according to WS-Distributed Management.

For the interoperability, the tools to manipulate schemas and WSDL contracts are still badly connect to generation tools Contract First strategy is difficult to implement. A problem is that most of developers write the code first and do not plan the contract first.

10 Thanks

We sincerely thank everyone who directly or indirecly has contributed and supported us in different ways to achieve our educational objectives.

We provide a special to KEYMILE for the perfect hosting of this project, specially to Daniel Gachet and Marcel Bellorini for the offered time.

The project sessions with Philippe Joye, François Buntschu, Marcel Bellorini and Daniel Gachet had a lot of very helpful and guiding inputs.

Another special big thank-you to all the proofreaders. Specially Hervé Kiki, Jörg Schneider and Tamara Eggimann.

11 Annexes

Abstract

In this last chapter you will find the references for this report and the revision history to comprehend the modifications on the document.

11.1 References

11.1.1 Keymile

- Introduction to the MileGate XML - Management Interface
- FILE: KEYMILE_XML-management-interface.pdf
- Web Services Interface for Milegate
- FILE: KEYMILE_WebServiceInterface.ppt
- User Guide – MileGate & MCST (KEYMILE Restriction)
- C++ Programming Style Guidelines, Common Part (KEYMILE Confidential)
- C++ Programming Practice Guidelines, Common Part (KEYMILE Confidential)

11.1.2 Service Oriented Architecture / Web Service Architecture

- W3C documents about Web Service Architecture
  http://www.w3.org/2002/ws/arch/
- Reference Model for Service Oriented Architecture 1.0
  http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf
- Article: What Is Service-Oriented Architecture on webservices.xml.com
  http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html
- Book: Service-orientierte Architekturen mit Web Services, Konzepte – Standards – Praxis, Ingo Melzer et al., SPEKTRUM Akademischer Verlag
- Book: Web Services. Principles and Technology, Michael P. Papazoglou, PEARSON

11.1.3 Webservice description / concepts

- WSDL 1.1, www.w3.org/TR/wsdl
- WSDL 2.0, www.w3.org/TR/wsdl20
- SOAP, W3C Recommendation,
  www.w3.org/TR/soap/

- SOAP 1.2, W3C Recommendation, http://www.w3.org/TR/soap12/
- XSLT 1.0, W3C Recommendation, http://www.w3.org/TR/xslt
- XSLT 2.0, W3C Recommendation, http://www.w3.org/TR/xslt20
- XHTML 1.0, W3C Recommendation, http://www.w3.org/TR/xhtml1/
- WS-Addressing, W3C Recommendation, http://www.w3.org/TR/ws-addr-core/
- WS-A: WSDL Binding, W3C Recommendation, http://www.w3.org/TR/ws-addr-wsdl/
- WS-A: SOAP Binding, W3C Recommendation, http://www.w3.org/TR/ws-addr-soap/
- WS-A: Metatdata, W3C Recommendation, http://www.w3.org/TR/ws-addr-metadata/
- WS-Management, Distributed Management Task Force, http://www.dmtf.org/standards/published_documents/DSP0226_1.0.0.pdf
- WS-Transfer, W3C Submission, http://www.w3.org/Submission/WS-Transfer/
- WS-Discovery, OASIS Committee Specification, http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.pdf
- WS-Base Notification, OASIS Standard, http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf
- WS-Brokered Notification, OASIS Standard, http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf
- WS-Topics, OASIS Standard, http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf
- WS-Resource, OASIS Standard, http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf

- WS-Resource Properties, OASIS Standard, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf
- WS-Distributed Management: Management Using Web Services MUWS Part 1, OASIS Standard, http://docs.oasis-open.org/wsdm/wsdm-muws1-1.1-spec-os-01.pdf
- WS-Distributed Management: Management Using Web Services MUWS Part 2, OASIS Standard http://docs.oasis-open.org/wsdm/wsdm-muws2-1.1-spec-os-01.pdf
- UDDI (Universal Description, Discovery and Integration), OASIS Standard http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf

11.1.4 Embedded Webserver

http://www.appwebserver.org/
http://www.lighttpd.net/
http://www.nginx.org/
http://www.emweb.be/
http://www.cherokee-project.com/
http://www.goahead.com/products/webserver/Default.aspx
http://www.koanlogic.com/klone/features.html
http://www.allegrosoft.com/rpproduct.html
http://barracudaserver.com/Barracuda_web_server_SDK.html
http://www.iniche.com/webport.php
http://news.netcraft.com/archives/web_server_survey.html (popularity)
http://googleonlinesecurity.blogspot.com/2007/06/web-server-software-and-malware.html (popularity)

11.1.5 Frameworks

AXIS2
- http://ws.apache.org/
- http://www.oasis-open.org/
- http://wso2.org/

CXF
- http://cxf.apache.org/

WSO2 Frameworks
- http://wso2.org/projects/wsf

- http://wso2.org/interop
- http://wso2.org/projects/commons

METRO

- https://wsit.dev.java.net/
- https://jaxb.dev.java.net/
- https://jax-ws.dev.java.net/
- http://java.sun.com/webservices/index.jsp

WCF

- http://msdn.microsoft.com/fr-fr/default.aspx

Others

- http://json.org/
- http://xmlbeans.apache.org/
- http://www.w3.org/TR/soap12-mtom/
- http://java.sun.com/developer/technicalArticles/xml/fastinfoset/

11.2 Figures

Illustration 10: Before & after SOA
Source: http://www.sun.com/products/soa/img/ig_soa_before_after.gif

Illustration 11: Three roles in SOA
Source: http://www.w3.org/2003/Talks/0521-hh-wsa/soa.png

Illustration 37, 42, 43
Source: http://msdn.microsoft.com

11.3 Complementary Information

Annexe:Journal

Annexe:PV

Annexe:Planing

Annexe:HTML Service

Annexe:WSDL

Annexe:XSLT